ARock: an Algorithmic Framework for Async-Parallel Coordinate Updates

Zhimin Peng · Yangyang Xu · Ming Yan · Wotao Yin

July 7, 2015

The problem of finding a fixed point to a nonexpansive operator is an abstraction of many models in numerical linear algebra, optimization, and other areas of scientific computing. To solve this problem, we propose ARock, an asynchronous parallel algorithmic framework, in which a set of agents (machines, processors, or cores) update randomly selected coordinates of the unknown variable in an asynchronous parallel fashion. The resulting algorithms are not affected by load imbalance. When the coordinate updates are atomic, the algorithms are free of memory locks.

We show that if the nonexpansive operator has a fixed point, then with probability one, the sequence of points generated by ARock converges to a fixed point of the operator. Stronger convergence properties such as linear convergence are obtained under stronger conditions. As special cases of ARock, novel algorithms for linear systems, convex optimization, machine learning, distributed and decentralized optimization are introduced with provable convergence. Very promising numerical performance of ARock has been observed. Considering the paper length, we present the numerical results of solving linear equations and sparse logistic regression problems.

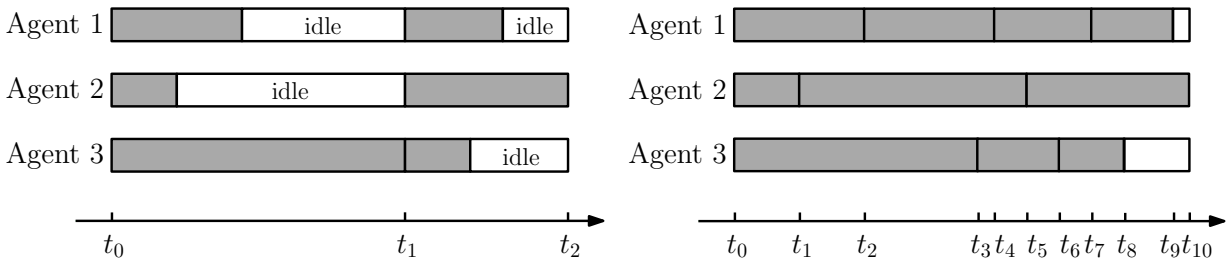
Contents

Z. Peng · M. Yan · W. Yin
Department of Mathematics, University of California, Los Angeles, CA 90095, USA
E-mail: zhimin.peng / yanm / wotaoyin@math.ucla.edu
Y. Xu
Department of Combinatorics and Optimization, University of Waterloo, Waterloo, ON N2L3G1, Canada
E-mail: yangyang.xu@uwaterloo.ca

1 Introduction

Technological advances in data gathering and storage have led to a rapid proliferation of big data in diverse areas such as climate studies, cosmology, medicine, the Internet, and engineering [31]. The data involved in many of these modern applications are so large and grow so quickly that new computational approaches that analyze these data in a parallel fashion are needed. This paper introduces an approach to asynchronous parallel computing.

Asynchrony reduces idle time of agents (machines, processors, or cores) and alleviates congestions in communication and memory access. When multiple agents execute a *synchronous* parallel (sync-parallel) iterative algorithm (Figure 1a), all the agents must wait until the slowest agent has finished its iteration before they can start the next iteration. Therefore, the slowest agent determines the global speed. In addition, in a sync-parallel algorithm, all the agents simultaneously make congestion-inducing communication and memory-access requests. When multiple agents work in an *asynchronous* parallel (async-parallel) fashion (Figure 1b), they run continuously and spread the congestion-inducing communication and memory-access contention.



(a) Sync-parallel computing (b) Async-parallel computing

Fig. 1: Sync-parallel computing (left) versus async-parallel computing (right). On the left, all the agents must wait at idle (white boxes) until the slowest agent has finished.

Asynchrony has other advantages (c.f. [10]): the system is more fault-tolerant since an asynchronous algorithm can continue to run after the failure of some agents; and it is easy to incorporate new agents.

However, it is more difficult to analyze asynchronous algorithms and ensure their convergence. There is no longer a sequence of iterations where the output of one iteration determines that of the next iteration. This paper proposes a novel async-parallel coordinate update framework for the generic fixed-point iteration, which has many algorithms as its special cases. The fixed-point operator only needs to be nonexpansive and have a fixed point.

Let $\mathcal{H}_1, \ldots, \mathcal{H}_m$ denote m Hilbert spaces, and let $\mathcal{H} := \mathcal{H}_1 \times \cdots \times \mathcal{H}_m$ be their Cartesian product. Our problem is to find a fixed point x^* to a *nonexpansive operator*[1] $T : \mathcal{H} \to \mathcal{H}$, that is,

$$\text{find } x^* \in \mathcal{H} \qquad \text{such that} \qquad x^* = Tx^*. \tag{1}$$

Note that finding a fixed point to T is equivalent to finding a zero of

$$S \equiv I - T,$$

[1] An operator $T : \mathcal{H} \to \mathcal{H}$ is nonexpansive if T is 1-Lipschitz, that is, $\|Tx - Ty\| \leq \|x - y\|$, $\forall x, y \in \mathcal{H}$.

that is, find x^* such that $0 = Sx^*$. Hereafter, we will use both S and T for convenience. Problem (1) is widely applicable in linear and nonlinear equations, statistical regression, machine learning, convex optimization, and optimal control. A generic framework for problem (1) is the Krasnosel'skiĭ–Mann (KM) iteration [34]:

$$x^{k+1} = x^k + \alpha\,(Tx^k - x^k) \quad \text{or} \quad x^{k+1} = x^k - \alpha S x^k, \tag{2}$$

where $\alpha \in (0,1)$ is the step size. If $\operatorname{Fix} T$ — the set of fixed points of T (zeros of S) — is nonempty, then the sequence $(x^k)_{k\geq 0}$ converges weakly[2] to a point in $\operatorname{Fix} T$ and $(Tx^k - x^k)_{k\geq 0}$ converges strongly to 0. The KM iteration abstracts algorithms in convex optimization, linear algebra, differential equations, and monotone inclusions. Its special cases include the following iterations: alternating projection, gradient descent, projected gradient descent, proximal-point algorithm, Forward-Backward Splitting (FBS) [43], Douglas-Rachford Splitting (DRS) [37], a three-operator splitting [20], and the Alternating Direction Method of Multipliers (ADMM) [37,29].

This paper introduces an algorithmic framework, ARock, where a set of p agents, $p \geq 1$, solves problem (1) by performing updates to the coordinates $x_i \in \mathcal{H}_i$, $i = 1,\dots,m$, in an independent, random, and asynchronous fashion. Algorithm 1 describes the framework. Its special forms for various applications are given in Section 2 below.

Algorithm 1: ARock: a framework for async-parallel coordinate updates

Input : $x^0 \in \mathcal{H}$, $K > 0$, a discrete distribution $(p_1,\dots,p_m)$ with $\sum_{i=1}^m p_i = 1$ and $p_i > 0, \forall i$,
set global iteration counter $k = 0$;
while $k < K$, *every agent asynchronously and continuously* **do**
  select $i_k \in \{1,\dots,m\}$ with $\text{Prob}(i_k = i) = p_i$;
  perform an update to x_{i_k} according to (3);
  update the global counter $k \leftarrow k + 1$;

Whenever an agent updates a coordinate, the global iteration number k increases by one. The kth update is applied to $x_{i_k} \in \mathcal{H}_{i_k}$, where $i_k \in \{1,\dots,m\}$ is an independent random variable. Each coordinate update has the form:

$$x^{k+1} = x^k - \frac{\eta_k}{m p_{i_k}} S_{i_k} \hat{x}^k, \tag{3}$$

where $\eta_k > 0$ is a scalar whose range will be set later,

$$S_{i_k} x := (0,...,0,(Sx)_{i_k},0,...,0),$$

mp_{i_k} is used to normalize nonuniform selection probabilities. In the uniform case, namely, $p_i \equiv \frac{1}{m}$ for all i, we have $mp_{i_k} \equiv 1$, which simplifies the update (3) to

$$x^{k+1} = x^k - \eta_k S_{i_k} \hat{x}^k. \tag{4}$$

Here, the point $\hat{x}^k$ is the result that x in global memory is read by an agent to its local cache and to which S_{i_k} is applied, and x^k denotes the state of x in global memory just before the update (3) is applied. In a sync-parallel algorithm, we have $\hat{x}^k = x^k$, but in ARock, due to possible updates to x by other agents, $\hat{x}^k$

[2] A sequence $(y^k)_{k\geq 0} \subset \mathcal{H}$ converges weakly to a point y if $\langle y^k, z\rangle \to \langle y, z\rangle$ for every $z \in \mathcal{H}$. A sequence $(y^k)_{k\geq 0} \subset \mathcal{H}$ converges strongly to a point y if $\|y^k - y\| \to 0$. Strong convergence gives weak convergence, but not vice versa. If $\mathcal{H}$ has a finite dimension, then they are equivalent.

can be different from x^k. This is a key difference between sync-parallel and async-parallel algorithms. In Subsection 1.2 below, we will establish the relationship between $\hat{x}^k$ and x^k as

$$\hat{x}^k = x^k + \sum_{d \in J(k)} (x^d - x^{d+1}), \tag{5}$$

where $J(k) \subseteq \{k-1, \ldots, k-\tau\}$, and $\tau \in \mathbb{Z}^+$ is the maximum number of other updates to x during the computation of (3). Equation (5) has appeared in [38].

The convergence of ARock (Algorithm 1) is stated in Theorems 3 and 4. Here we include a shortened version:

Theorem 1 (Global and linear convergence) *Let $T : \mathcal{H} \to \mathcal{H}$ be a nonexpansive operator with a nonempty set of fixed points. Let $(x^k)_{k\geq 0}$ be the sequence generated by Algorithm 1 with properly bounded step sizes η_k. Then, with probability one, $(x^k)_{k\geq 0}$ converges weakly to a fixed point of T. This weak convergence is automatically improved to strong convergence if $\mathcal{H}$ has a finite dimension.*

In addition, if T is demicompact (see Definition 2 below), then $(x^k)_{k\geq 0}$ converges strongly to a fixed point of T with probability one.

Furthermore, if $S \equiv I - T$ is quasi-strongly monotone (see Definition 1 below), then T has a unique fixed-point x^*, $(x^k)_{k\geq 0}$ converges strongly to x^*, and $\mathbb{E}\|x^k - x^*\|^2$ converges to 0 at a linear rate.

Note that we only impose the standard assumption on T that it is nonexpansive and has a fixed point. We have made the following assumptions to the computation: (a) bounded step sizes; (b) random coordinate selection; and (c) a finite maximal delay τ. Assumption (a) is standard. Assumption (b) is essential to both the analysis and the strong numerical performance of our algorithms. Assumption (c) is *not* essential; an infinite delay with a light tail is allowed (but we leave it to future work).

1.1 On random coordinate selection

ARock employs random coordinate selection. This subsection discusses its advantages and disadvantages.

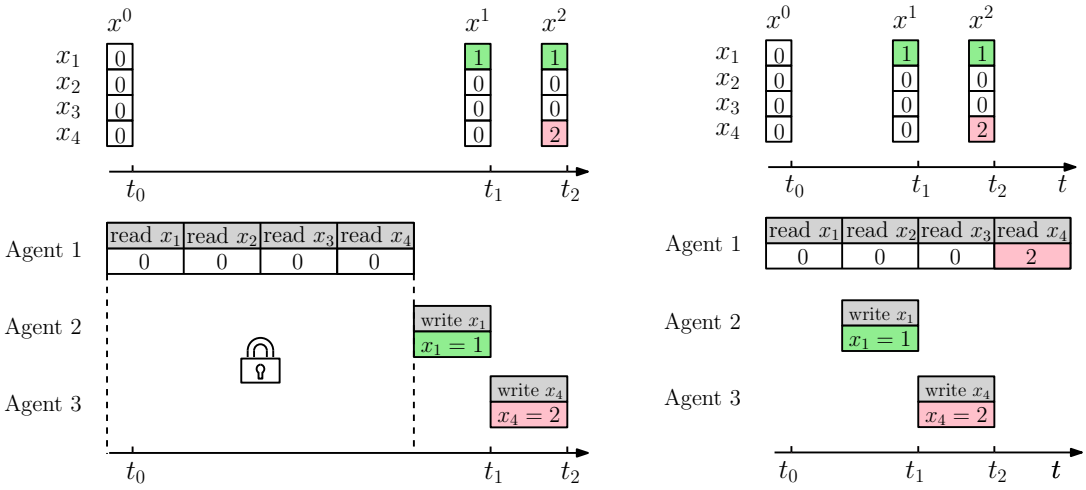
Its main disadvantage is that it prevents an agent from caching a coordinate and its related data. Therefore, in general, the variable x and its related data must be either stored in global memory or passed through a network during random coordinate assignments. A secondary disadvantage is that pseudo-random number generation requires time, which can become relatively significant when each coordinate update is cheap. (The network optimization examples in Subsection 2.3 and Subsection 2.6.2 are exceptions, where storage is local and random coordinate assignments are not generated but the results of events follow Poisson processes.)

There are several advantages of random coordinate selection. It realizes the user-specified update frequency p_i for every component x_i, $i = 1, \ldots, m$, even when different agents have different computing powers and/or different coordinate updates cost different amounts of computation. Therefore, random assignment ensures load balance and fault tolerance. In addition, it has been observed numerically on certain problems [14] that random coordinate selection accelerates convergence. Furthermore, in [54,55], the algorithms with random coordinate selection keep clear of low-quality local solutions in nonconvex optimization, which are difficult to avoid by fixed-order coordinate selection.

1.2 Uncoordinated memory access

In ARock, since multiple agents may simultaneously read and update x in global memory, $\hat{x}^k$ — the result of x that is read from global memory by an agent to its local cache for computation — may not equal x^j for any $j \leq k$, that is, $\hat{x}^k$ may never be consistent with a state of x in global memory. This situation is known as *inconsistent read*. In contrast, *consistent read* means that $\hat{x}^k = x^j$ for some $j \leq k$, i.e., $\hat{x}^k$ is consistent with a state of x that existed in global memory.

We illustrate inconsistent read and consistent read in the following example, which is depicted in Figure 2. Consider $x = [x_1, x_2, x_3, x_4]^T \in \mathbb{R}^4$ and $x^0 = [0, 0, 0, 0]^T$ initially, at time t_0. Suppose at time t_1, agent 2 updates x_1 from 0 to 1, yielding $x^1 = [1, 0, 0, 0]^T$; then, at time t_2, agent 3 updates x_4 from 0 to 2, further yielding $x^2 = [1, 0, 0, 2]^T$. Suppose that agent 1 starts reading x from the first component x_1 at t_0. For consistent read (Figure 2a), agent 1 acquires a memory lock and only releases the lock after finishing reading all of x_1, x_2, x_3, and x_4. Therefore, agent 1 will read in $[0, 0, 0, 0]^T$. Inconsistent read, however, allows agent 1 to proceed without a memory lock: agent 1 starts reading x_1 at t_0 (Figure 2b) and reaches the last component, x_4, after t_2; since x_4 is updated by agent 3 prior to it is read by agent 1, agent 1 has read $[0, 0, 0, 2]^T$, which is different from any of x^0, x^1, x^2. Even with inconsistent read, each component



(a) Consistent read. While agent 1 reads x in memory, it acquires a global lock.

(b) Inconsistent read. No lock. Agent 1 reads $(0, 0, 0, 2)^T$, a non-existing state of x.

Fig. 2: Consistent read versus inconsistent read: A demonstration.

is consistent (under the *atomic coordinate update* assumption, which will be defined below.) Therefore, we can express the result of reading using the changes of individual coordinates. In the above example, the first change is $x_1^1 - x_1^0 = 1$, which is added to x_1 just before time t_1 by agent 2, and the second change is $x_4^2 - x_4^1 = 2$, added to x_4 just before time t_2 by agent 3. The inconsistent read by agent 1, which gives the result $[0, 0, 0, 2]^T$, equals $x^0 + 0 \times (x^1 - x^0) + 1 \times (x^2 - x^1)$.

We have demonstrated that $\hat{x}^k$ can be inconsistent, but each of its coordinates is consistent, that is, for each i, $\hat{x}_i^k$ is an ever-existed state of x_i among $x_i^k, \ldots, x_i^{k-\tau}$. Suppose that $\hat{x}_i^k = x_i^{\underline{d}}$, where $\underline{d} \in \{k, k-1, \ldots, k-$

$\tau\}$. Therefore, $\hat{x}_i^k$ can be related to x_i^k through the *interim changes* applied to x_i. Let $J_i(k) \subset \{k-1, \ldots, k-\tau\}$ be the index set of these interim changes. If $J_i(k) \neq \emptyset$, then $\underline{d} = \min\{d \in J_i(k)\}$; otherwise, $\underline{d} = k$. In addition, we have

$$\hat{x}_i^k = x_i^{\underline{d}} = x_i^k + \sum_{d \in J_i(k)} (x_i^d - x_i^{d+1}). \tag{6}$$

Since the global counter k is increased after each coordinate update, updates to x_i and x_j, $i \neq j$, must occur at different k's and thus $J_i(k) \cap J_j(k) = \emptyset$, $\forall i \neq j$. Therefore, by letting $J(k) := \cup_i J_i(k) \subset \{k-1, \ldots, k-\tau\}$ and noticing $(x_i^d - x_i^{d+1}) = 0$ for $d \in J_j(k)$ where $i \neq j$, we have

$$\hat{x}_i^k = x_i^k + \sum_{d \in J(k)} (x_i^d - x_i^{d+1}), \quad \forall i = 1, \ldots, m,$$

which is equivalent to (5). Here, we have made two assumptions:

- *atomic coordinate update*: a coordinate is not further broken to smaller components during an update; they are all updated at once.
- *bounded maximal delay* τ: during any update cycle of an agent, x in global memory is updated at most τ times by other agents.

When each coordinate is a single scalar, updating the scalar is a single atomic instruction on most modern hardware, so the first assumption naturally holds, and our algorithm is *lock-free*. The case where a coordinate includes multiple scalars is discussed in the next subsection.

1.2.1 Block coordinate

When each coordinate involves more than one scalar component, i.e., in the "block coordinate" case, we satisfy the atomic coordinate update assumption by *either* employing a per-coordinate memory lock *or* taking the dual-memory approach as follows.

The dual-memory approach: Store *two* copies of each coordinate $x_i \in \mathcal{H}_i$ in global memory, denoting them as $x_i^{(0)}$ and $x_i^{(1)}$, and use a bit $\alpha_i \in \{0, 1\}$ to point to the active copy. An agent will only read x_i from the active copy $x_i^{(\alpha_i)}$. When an agent starts to update the components of x_i, it first obtains a memory lock to the *inactive* copy $x_i^{(1-\alpha_i)}$ to prevent other agents from simultaneously updating it, then applies the update to $x_i^{(1-\alpha_i)}$, and finally flips the bit α_i so that other agents will begin reading the newly updated copy. This approach never blocks any read of x_i, yet it completely eliminates inconsistent read of x_i.

1.3 Straightforward generalization: orthogonal update

While our async-parallel iteration (3) is updated in a coordinate fashion, the scheme can be directly extended to orthogonal updates, and the convergence analysis continues to hold. More specifically, we can perform the following update

$$x^{k+1} = x^k - \frac{\eta_k}{m p_{i_k}} (U_{i_k} \circ S)\hat{x}^k, \tag{7}$$

where the operator U_{i_k} is randomly drawn from a set of *orthogonal* operators $\{U_1, \ldots, U_m\}$, $U_i : \mathcal{H} \to \mathcal{H}$, that is, $\langle U_i(x), U_j(x)\rangle = 0$ for all $i \neq j$, following the probability $P(i_k = i) = p_i$, $i = 1, \ldots, m$. An example of orthogonal operators is $U_i : x \mapsto (0, \ldots, x_i, \ldots, 0)$, $i = 1, \ldots m$, which reduces (7) to (3). Another example is $U_i : x \mapsto x_i \mathbf{a}_i$, $i = 1, \ldots m$, where $\{\mathbf{a}_i\}$ is a set of orthogonal entries of $\mathcal{H}$.

1.4 Special cases

If there is only one agent ($p = 1$), ARock (Algorithm 1) reduces to randomized coordinate update, which includes the special case of randomized coordinate descent [42] for convex optimization. ARock also includes sync-parallel coordinate update as a special case where $\hat{x}^k = x^k$, $\forall k$. In both cases, there is no delay, i.e., $\tau = 0$ and $J(k) = \emptyset$. In addition, the step size η_k can take more relaxed values. In particular, if $p_i = \frac{1}{m}$, $\forall i$, then we can let $\eta_k = \eta$, $\forall k$, for any $\eta < 1$ (or $\eta < 1/\alpha$ when T is α-averaged).

1.5 Related work

Chazan and Miranker [16] proposed the first async-parallel method in 1969. The method was designed for solving linear systems. Async-parallel methods have been successful in many fields, e.g., linear systems [3, 11, 26, 48], nonlinear problems [4, 5], differential equations [1, 2, 15, 21], consensus problems [36, 25], and optimization [32, 38, 39, 50, 58]. We review the theory for async-parallel fixed-point iteration and its applications.

General fixed point problems. *Totally async-parallel*[3] iterative methods for a general fixed-point problem go back as early as to Baudet [5]. The operator in [5] was assumed to be *Absolute Lipschitz Contractive* (ALC).[4] Later, Bertsekas [8] generalized the ALC assumption and showed convergence. Bahi, Miellou, and Rhofir [4] combined the results in [5] with the so-called multisplittling method. Papers [7, 22] provide a convergence result for async-parallel iterations with simultaneous reading and writing of the same set of components. Strikwerda considered the unbounded but stochastic delays [49]. Frommer and Szyld [27] reviewed the theory and applications of totally async-parallel iterations through year 2000. This review summarized convergence results under the conditions in [8].

Tseng, Bertsekas, and Tsitsiklis [9, 52] assumed *quasi-nonexpansiveness*[5] and proposed an async-parallel method, which was shown to converge under an additional difficult-to-justify assumption.

Linear, nonlinear, and differential equations. The first totally async-parallel method for solving linear equations was introduced by Chazan and Miranker in [16]. They proved for linear systems that ALC was necessary and sufficient. The performance of the algorithm was studied by Iain et al. [11, 48] on different High Performance Computing (HPC) architectures. Frommer, Schwandt, and Szyld [26] proposed a class of Schwarz methods. Recently, Avron et al. [3] proposed an async-parallel coordinate update for solving a linear system with a symmetric positive-definite matrix. Tarazi and Nabih [24] extended the work by [16] to solving nonlinear equations. Applications of async-parallel methods for solving differential equations can be found in [1, 2, 15, 21].

Optimization. The first async-parallel coordinate update gradient-projection method was due to Bertsekas and Tsitsiklis [9]. The method solves constrained optimization problems with a smooth objective and simple constraints. It was shown that the objective gradient sequence converges to zero. Tseng [51] further analyzed the convergence rate and obtained local linear convergence based on the assumptions of isocost surface separation and a local Lipschitz error bound. Recently, Liu et al. [39] developed an async-parallel stochastic coordinate descent algorithm for minimizing convex smooth functions. Later, Liu and Wright [38] suggested an async-parallel stochastic proximal coordinate descent algorithm for minimizing convex composite objective functions. Assuming step size decaying exponentially in the maximum delay, they established

[3] "Totally asynchronous" means no upper bound on the delays; however, each coordinate must be updated infinitely many times. By default, "asynchronous" in this paper refers to having a finite maximum delay.

[4] An operator $T : \mathbb{R}^n \to \mathbb{R}^n$ is max-norm contractive if $|T(x) - T(y)| \leq A|x - y|$, component-wise, where $|x|$ denotes the vector with components $|x_i|$, $i = 1, ..., n$, and $A \in \mathbb{R}^{n\times n}$ is a matrix with a spectral radius strictly less than 1.

[5] An operator $T : \mathcal{H} \to \mathcal{H}$ is quasi-nonexpansive if $\|Tx - x^*\| \leq \|x - x^*\|$, $\forall x \in \mathcal{H}$, $x^* \in \operatorname{Fix} T$.

the convergence of the expected objective-error sequence for convex functions. Hsieh et al. [32] proposed an async-parallel implementation of LIBLINEAR (a library for solving ℓ_2 regularized empirical risk minimization problems). Other async-parallel approaches include asynchronous ADMM [30,53,58,33]. Among them, [53,33] use an asynchronous clock, and [30,58] use a central node to update the dual variable; they do not deal with delay or inconsistency. Async-parallel stochastic gradient descent methods have also been considered in [40,45].

Our framework differs from the recent surge of the aforementioned sync-parallel and async-parallel coordinate descent algorithms (e.g., [44,35,39,38,32,46]). While they are based on minimizing convex functions, ARock aims at finding a fixed point to a nonexpansive operator, and our iterations decrease the expected distance to the fixed point set. In Section 2, we will show that some of the existing async-parallel coordinate descent algorithms are special cases of our framework, through relating the optimality conditions of their minimization problems to nonexpansive operators. Another difference is that the convergence of ARock only requires the operator to be nonexpansive and to have a fixed point, whereas properties such as strong convexity, bounded feasible set, bounded sequence that are not necessary for the convergence in standard convex minimization are frequently seen in the recent literature to circumvent certain technical difficulties.

Others. Besides solving equations and optimization problems, there are also applications of async-parallel algorithms to optimal control problems [36], network flow problems [23], and consensus problems of multi-agent systems [25].

1.6 Contributions

We now describe our contributions and techniques as follows:

- We introduce ARock, the first async-parallel coordinate update framework, for the Krasnosel'skiĭ–Mann (KM) iteration.
- Our convergence analysis only makes the standard fixed-point assumption: a nonexpansive operator with a fixed point. Under a new metric, we show stochastic Fejér monotonicity and that the ARock sequence weakly converges, almost surely, to a $(\operatorname{Fix} T)$-valued random variable; strong convergence is obtained under slightly stronger conditions; linear convergence is proven for *quasi-strongly monotone* (See Definition 1) operators.
- Using ARock, we develop several new async-parallel algorithms including a new async-parallel scheme for solving linear equations, new async-parallel ADMM algorithms for solving consensus or decentralized optimization problems, a new async-parallel forward-backward splitting method, and a new async-parallel Peaceman-Rachford splitting method for minimizing composite nonsmooth functions. Some of these async-parallel algorithms are *not* straightforward modifications to their sequential versions. They must be obtained by applying ARock along with sophisticated operator splitting.

1.7 Notation, definitions, background of monotone operators

Throughout this paper, $\mathcal{H}$ denotes a separable Hilbert space equipped with the inner product $\langle \cdot, \cdot \rangle$ and norm $\| \cdot \|$, and $(\Omega, \mathcal{F}, P)$ denotes the underlying probability space, where Ω, $\mathcal{F}$, and P are the sample space, σ-algebra, and probability measure, respectively. The map $x : (\Omega, \mathcal{F}) \to (\mathcal{H}, \mathcal{B})$, where $\mathcal{B}$ is the Borel σ-algebra, is an $\mathcal{H}$-valued random variable. Let $(x^k)_{k\geq 0}$ denote *either* a sequence of deterministic points in $\mathcal{H}$ *or* a sequence of $\mathcal{H}$-valued random variables, which will be clear from the context, and let $x_i \in \mathcal{H}_i$ denote

the ith coordinate of x. In addition, we let $\mathcal{X}^k := \sigma(x^0, x^1, ..., x^k)$ denote the smallest σ-algebra generated by $x^0, x^1, ..., x^k$. "Almost surely" is abbreviated as "a.s.", and the n product space of $\mathcal{H}$ is denoted by $\mathcal{H}^n$. We use $\to$ and $\rightharpoonup$ for strong convergence and weak convergence, respectively.

We define $\operatorname{Fix} T := \{x \in \mathcal{H} \mid Tx = x\}$ as the set of the fixed points of the operator T and, in the product space, we let $\mathbf{X}^* := \{(x^*, x^*, ..., x^*) \mid x^* \in \operatorname{Fix} T\} \subseteq \mathcal{H}^{\tau+1}$.

Definition 1 An operator $T : \mathcal{H} \to \mathcal{H}$ is *c-Lipschitz*, where $c \geq 0$, if it satisfies $\|Tx - Ty\| \leq c\|x - y\|$, $\forall x, y \in \mathcal{H}$. In particular, T is *nonexpansive* if $c \leq 1$, and *contractive* if $c < 1$.

Definition 2 Consider an operator $T : \mathcal{H} \to \mathcal{H}$.

- T is *α-averaged*, where $\alpha \in (0, 1)$, if there exists a nonexpansive operator $R : \mathcal{H} \to \mathcal{H}$ such that $T = (1 - \alpha)I_{\mathcal{H}} + \alpha R$, where $I_{\mathcal{H}}$ is the identity map from $\mathcal{H}$ to $\mathcal{H}$.
- T is *β-cocoercive*, where $\beta > 0$, if it satisfies $\langle x - y, Tx - Ty\rangle \geq \beta\|Tx - Ty\|^2,\ \forall x, y \in \mathcal{H}$.
- T is *μ-strongly monotone*, where $\mu > 0$, if it satisfies $\langle x - y, Tx - Ty\rangle \geq \mu\|x - y\|^2,\ \forall x, y \in \mathcal{H}$. When the inequality holds for $\mu = 0$, T is *monotone*.
- T is *quasi-μ-strongly monotone*, where $\mu > 0$, if it satisfies $\langle x - y, Tx\rangle \geq \mu\|x - y\|^2,\ \forall x \in \mathcal{H}, y \in \operatorname{zer} T := \{y \in \mathcal{H} \mid Ty = 0\}$. When the inequality holds for $\mu = 0$, T is *quasi-monotone*.
- T is *demicompact at $x \in \mathcal{H}$* if for every bounded sequence $(x^k)_{k\geq 0}$ in $\mathcal{H}$ such that $Tx^k - x^k \to x$, there exists a strongly convergent subsequence.

All averaged operators are nonexpansive. By the Cauchy-Schwarz inequality, any β-cocoercive operator is $\frac{1}{\beta}$-Lipschitz, and the converse is generally untrue, but true for the gradients of convex differentiable functions. Examples of operators are given in the next section.

2 Motivation and Applications

Our async-parallel framework ARock generates simple numerical schemes to a large number of problems that appear in scientific computing, signal processing, machine learning, sensor network, and other data science areas. In this section, we provide some applications that are special cases of the fixed-point problem (1). For each application, we briefly mention how to satisfy the conditions for Theorem 1. Some parameters for each application are not optimal, but we do not pursue it here.

For simplicity, we assume uniform distribution, namely, $p_1 = p_2 = \cdots = p_m = 1/m$, and apply the simpler update (4) instead of (3). All of the algorithms in this section can take non-uniform distributions.

2.1 Solving linear equations

The problem is to find a solution to the following linear system

$$Ax = b, \tag{8}$$

where $A \in \mathbb{R}^{m\times m}$ is a nonsingular matrix with nonzero diagonal entries. Let $A = D + R$, where D and R are the diagonal and off-diagonal parts of A, respectively. Let $M := -D^{-1}R$ and $T(x) := Mx + D^{-1}b$. Then the system (8) is equivalent to the fixed-point equation

$$x = D^{-1}(b - Rx) =: T(x). \tag{9}$$

Note T is nonexpansive if the spectral norm $\|M\|_2$ of M satisfies $\|M\|_2 \leq 1$. The iteration $x^{k+1} = T(x^k)$ is known as the Jacobi algorithm. Let $S = I - T$. When applying our algorithm, each update $S_{i_k}\hat{x}^k$ involves multiplying just the i_kth row of M to x and adding the i_kth entry of $D^{-1}b$, so we have the following algorithm for solving linear equations.

Algorithm 2: ARock for linear equations

Input : $x^0 \in \mathbb{R}^n$, $K > 0$;
set global iteration counter $k = 0$;
while $k < K$, *every agent asynchronously and continuously* **do**
  select $i_k \in \{1, \ldots, m\}$ uniformly at random;
  subtract $\frac{\eta_k}{a_{i_k i_k}}(\sum_j a_{i_k j}\hat{x}_j^k - b_{i_k})$ from the component x_{i_k} of the variable x;
  update the global counter $k \leftarrow k + 1$;

Proposition 1 *[6, Example 22.5] Suppose that T is c-Lipschitz continuous with $c \in [0, 1)$. Then, $I - T$ is $(1 - c)$-strongly monotone.*

Suppose $\|M\|_2 < 1$. Since T is $\|M\|_2$-Lipschitz continuous, by Proposition 1, S is $(1 - \|M\|_2)$-strongly monotone. Further by Theorem 4, Algorithm 2 converges linearly.

2.2 Minimizing convex smooth functions

Consider the following optimization problem

$$\underset{x\in\mathcal{H}}{\text{minimize}}\, f(x), \tag{10}$$

where f is a differentiable and closed proper convex function and ∇f is L-Lipschitz continuous, $L > 0$. Let $S := \frac{2}{L}\nabla f$. Since f is smooth and convex, x is a minimizer of $f(x)$ if and only if x is a zero of S. Note that S is $\frac{1}{2}$-cocoercive. By Lemma 1, $T \equiv I - S$ is nonexpansive. Applying ARock, we have the following iteration:

$$x^{k+1} = x^k - \eta_k S_{i_k}\hat{x}^k, \tag{11}$$

where $S_{i_k}x = \frac{2}{L}(0, ..., 0, \nabla_{i_k} f(x), 0, ..., 0)^T$. Note that ∇f should have a structure that makes it easier to compute $\nabla_{i_k} f(\hat{x}^k)$ than to evaluate the entire vector $\nabla f(\hat{x}^k)$. Otherwise, one would rather compute $\nabla f(\hat{x}^k)$ and perform the standard gradient update. Let us give two examples where $\nabla_i f(x)$ is much easier to compute than $\nabla f(x)$: (i) quadratic programming with $f(x) = \frac{1}{2}x^T Ax - b^T x$, where $\nabla f(x) = Ax - b$ and $\nabla_{i_k} f(\hat{x}^k)$ depends on a part of A and b; (ii) sum of sparsely supported functions with $f = \sum_{j=1}^N f_j$ and $\nabla f = \sum_{j=1}^N \nabla f_j$, where each f_j depends on just a few variables.

Theorem 3 guarantees the convergence of $(x^k)_{k\geq 0}$ if $\eta_k \in [\epsilon, \frac{1}{2\tau/\sqrt{m}+1})$. In addition, If $f(x)$ is *restricted strongly convex*, namely, for any $x \in \mathcal{H}$ and $x^* \in X^*$, where X^* is the solution set to (10), we have

$$\langle x - x^*, \nabla f(x) - \nabla f(x^*)\rangle \geq \mu\|x - x^*\|^2$$

for some $\mu > 0$, then S is quasi-strongly monotone with modulus μ. According to Theorem 4, iteration (11) converges at a linear rate if the step size meets the condition therein.

Our convergence results and rates are provided in terms of solution error. In comparison, the results in the recent work [39] are given in terms of objective error under the assumption that $(x^k)_{k\geq 0}$ is uniformly

bounded. In addition, their step size is required to decay like $O(\frac{1}{\tau\rho^\tau})$ for some $\rho > 1$, whereas ours is like $O(\frac{1}{\tau})$. Under similar assumptions, Bertsekas and Tsitsiklis [9, Section 7.5] proposed an algorithm for solving (10), for which only subsequence convergence [9, Proposition 5.3] in $\mathbb{R}^n$ is established.

2.3 Decentralized consensus optimization

Consider that n agents in a connected communication network collaboratively solve the consensus problem of minimizing $\sum_{i=1}^m f_i(x)$, where $x \in \mathbb{R}^d$ is the common variable and the convex differentiable function f_i is held privately by agent i. We assume that ∇f_i is L_i-Lipschitz continuous for all i. A decentralized gradient descent algorithm [41] can be developed based on the equivalent formulation

$$\begin{aligned} &\underset{x_1,\dots,x_m\in\mathbb{R}^d}{\text{minimize}} \;\; f(\mathbf{x}) := \textstyle\sum_{i=1}^m f_i(x_i) \\ &\text{subject to } W\mathbf{x} = \mathbf{x}, \end{aligned} \tag{12}$$

where $\mathbf{x} = (x_1, ..., x_m)^T \in \mathbb{R}^{m\times d}$ and $W \in \mathbb{R}^{m\times m}$ is the so-called mixing matrix such that $W\mathbf{x} = \mathbf{x}$ if and only if $x_1 = \cdots = x_m$, namely, the x_i's are consensual. If $w_{i,j} \neq 0$, $i \neq j$, then agent i can communicate with agent j. We assume that W is symmetric and doubly stochastic. Then, the decentralized consensus algorithm [41] can be expressed as $\mathbf{x}^{k+1} = W\mathbf{x}^k - \gamma\nabla f(\mathbf{x}^k) = \mathbf{x}^k - \gamma(\nabla f(\mathbf{x}^k) + \frac{1}{\gamma}(I - W)\mathbf{x}^k)$, where $\nabla f(\mathbf{x}) \in \mathbb{R}^{m\times d}$ is a matrix with its ith row equal to $(\nabla f_i(x_i))^T$; see [57]. The computation of $W\mathbf{x}^k$ involves communication between agents, and $\nabla f_i(x_i)$ is independently computed by each agent i. The iteration is equivalent to the gradient descent iteration applied to

$$\underset{x_1,\dots,x_m\in\mathbb{R}^d}{\text{minimize}} \;\; F(\mathbf{x}) := \textstyle\sum_{i=1}^m f_i(x_i) + \frac{1}{2\gamma}\mathbf{x}^T(I - W)\mathbf{x}. \tag{13}$$

To apply our algorithm, we let $S := \frac{2}{L}\nabla F = \frac{2}{L}(\nabla f + \frac{1}{\gamma}(I - W))$ with $L = \max_i L_i + (1 - \lambda_{\min}(W))/\gamma$, where $\lambda_{\min}(A)$ is the smallest eigenvalue of W. Computing $S_i\hat{\mathbf{x}}^k$ reduces to computing $\nabla f_i(\hat{x}_i^k)$ and the ith entry of $W\hat{\mathbf{x}}^k$ or $\sum_j w_{i,j}\hat{x}_j^k$, which involve only $\hat{x}_i^k$ and the neighbors of agent i. Note that since each agent i can store its own x_i locally, we have $\hat{x}_i^k \equiv x_i^k$.

To apply ARock, we assume that the agents are randomly activated following independent Poisson processes and that each agent i has access to $\hat{x}_j^k$ of its neighboring agents j up to a bounded delay (due to communication or system design). The agents can be otherwise uncoordinated. Under the independent Poisson assumption, the activation of agents is Markovian, and therefore our random sample scheme applies. The algorithm is summarized as follows:

Algorithm 3: ARock for decentralized optimization (12)

Input : Each agent i sets $x_i^0 \in \mathbb{R}^d$. A global stopping counter K.
while $k < K$ **do**
  when an agent i is activated, $x_i^{k+1} = x_i^k - \eta_k\left(\nabla f_i(x_i^k) + \frac{1}{\gamma}x_i^k - \sum_j w_{i,j}\,\hat{x}_j^k\right)$;
  increase the global counter $k \leftarrow k + 1$.

2.4 Minimizing the sum of smooth and nonsmooth functions

Consider the following problem:

$$\underset{x\in\mathcal{H}}{\text{minimize}}\, f(x)+g(x), \tag{14}$$

where f is closed proper convex and g is convex and L-Lipschitz differentiable with $L>0$. Problems in the form of (14) arise in statistical regression, machine learning, and signal processing and include well-known problems such as the support vector machine, regularized least-squares, and regularized logistic regression. For any $x\in\mathcal{H}$ and scalar $\gamma\in(0,\frac{2}{L})$, define the proximal operator $\mathbf{prox}_f:\mathcal{H}\to\mathcal{H}$ and the reflective-proximal operator $\mathbf{refl}_f:\mathcal{H}\to\mathcal{H}$ as

$$\mathbf{prox}_{\gamma f}(x):=\underset{y\in\mathcal{H}}{\arg\min}\, f(y)+\frac{1}{2\gamma}\|y-x\|^2 \quad \text{and} \quad \mathbf{refl}_{\gamma f}:=2\mathbf{prox}_{\gamma f}-I_{\mathcal{H}}, \tag{15}$$

respectively, and define the following forward-backward operator

$$T_{\text{FBS}}:=\mathbf{prox}_{\gamma f}\circ(I-\gamma\nabla g).$$

Because $\mathbf{prox}_{\gamma f}$ is $\frac{1}{2}$-averaged and $(I-\gamma\nabla g)$ is $\frac{\gamma L}{2}$-averaged, T_{FBS} is α-averaged for $\alpha\in[\frac{2}{3},1)$ [6, Props. 4.32 and 4.33]. Define $S:=I-T_{\text{FBS}}=I-\mathbf{prox}_{\gamma f}\circ(I-\gamma\nabla g)$. When we apply Algorithm 1 to $T=T_{\text{FBS}}$ to solve (14), and assume f is separable in all coordinates, that is, $f(x)=\sum_{i=1}^m f_i(x_i)$, the update for the i_kth selected coordinate is

$$x_{i_k}^{k+1}=x_{i_k}^k-\eta_k\left(\hat{x}_{i_k}^k-\mathbf{prox}_{\gamma f_{i_k}}(\hat{x}_{i_k}^k-\gamma\nabla_{i_k}g(\hat{x}^k))\right), \tag{16}$$

Examples of separable functions include ℓ_1 norm, ℓ_2 norm square, the Huber function, and the indicate function of box constraints, i.e., $\mathcal{I}_{\{x|a\le x\le b\}}(x)$. They all have simple **prox**. If $\eta_k\in[\epsilon,\frac{1}{2\tau/\sqrt{m}+1})$, then the convergence is guaranteed by Theorem 3. To show linear convergence, we need to assume that $g(x)$ is restricted strongly convex. Then, Proposition 2 below shows that $\mathbf{prox}_{\gamma f}\circ(I-\gamma\nabla g)$ is a quasi-contractive operator, and by Proposition 1, operator $I-\mathbf{prox}_{\gamma f}\circ(I-\gamma\nabla g)$ is quasi-strongly monotone. Finally, linear convergence and its rate follow from Theorem 4.

Proposition 2 *Assume that f is a closed proper convex function, and g is L-Lipschitz differentiable and restricted strongly convex with modulus $\mu>0$. Let $\gamma\in(0,\frac{2}{L})$. Both $I-\gamma\nabla g$ and $\mathbf{prox}_{\gamma f}\circ(I-\gamma\nabla g)$ are quasi-contractive operators.*

Proof We first show that $I-\gamma\nabla g$ is a contraction operator.

$$\begin{aligned}
&\|(x-\gamma\nabla g(x))-(x^*-\gamma\nabla g(x^*))\|^2\\
=&\|x-x^*\|^2-2\gamma\langle x-x^*,\nabla g(x)-\nabla g(x^*)\rangle+\gamma^2\|\nabla g(x)-\nabla g(x^*)\|^2\\
\le&\|x-x^*\|^2-2\gamma\langle x-x^*,\nabla g(x)-\nabla g(x^*)\rangle+\gamma^2L\langle x-x^*,\nabla g(x)-\nabla g(x^*)\rangle\\
\le&(1-2\gamma\mu+\mu\gamma^2L)\|x-x^*\|^2,
\end{aligned}$$

where the first inequality follows from the Baillon-Haddad theorem[6]. Hence, we have

$$\|(x-\gamma\nabla g(x))-(x^*-\gamma\nabla g(x^*))\|\le\sqrt{1-2\gamma\mu+\mu\gamma^2L}\,\|x-x^*\|,$$

[6] Let g be a convex differentiable function. Then, ∇g is L-Lipschitz if and only if it is $\frac{1}{L}$-cocoercive.

which means that $I-\gamma\nabla g$ is a quasi-contractive operator. Since f is convex, we know that $\mathbf{prox}_{\gamma f}$ is firmly nonexpansive, hence we have

$$\begin{aligned}&\|\mathbf{prox}_{\gamma f}\circ(I-\gamma\nabla g)(x)-\mathbf{prox}_{\gamma f}\circ(I-\gamma\nabla g)(x^*)\|\\ \le&\|(x-\gamma\nabla g(x))-(x^*-\gamma\nabla g(x^*))\|\\ \le&\sqrt{1-2\gamma\mu+\mu\gamma^2L}\,\|x-x^*\|.\end{aligned}$$

Hence, $\mathbf{prox}_{\gamma f}\circ(I-\gamma\nabla g)$ is a quasi-contractive operator.

2.5 Minimizing the sum of two nonsmooth functions

Consider the following problem

$$\underset{x\in\mathcal{H}}{\text{minimize}}\, f(x)+g(x), \tag{17}$$

where both $f(x)$ and $g(x)$ are closed proper convex and their **prox** maps are easy to compute. Define the Peaceman-Rachford [37] operator:

$$T_{\text{PRS}} := \mathbf{refl}_{\gamma f}\circ\mathbf{refl}_{\gamma g}.$$

Since both $\mathbf{refl}_{\gamma f}$ and $\mathbf{refl}_{\gamma g}$ are nonexpansive, their composition T_{PRS} is also nonexpansive. Let $S := I-T_{\text{PRS}}$. When applying ARock to $T=T_{\text{PRS}}$ to solve problem (17), the update (7) reduces to:

$$z^{k+1}=z^k-\eta_k U_{i_k}\circ\big(I-\mathbf{refl}_{\gamma f}\circ\mathbf{refl}_{\gamma g}\big)\hat{z}^k, \tag{18}$$

where we use z instead of x since the limit z^* of $(z^k)_{k\ge0}$ is not a solution to (17); instead, a solution to (17) is recovered via $x^*=\mathbf{prox}_{\gamma g}z^*$. The convergence follows from Theorem 3 and that T_{PRS} is nonexpansive. If either f or g is strongly convex, then T_{PRS} is contractive and thus by Theorem 4, ARock converges linearly. Finer convergence properties follow from [18,19]. A naive implementation of (18) is

$$\hat{x}^k=\mathbf{prox}_{\gamma g}(\hat{z}^k), \tag{19a}$$

$$\hat{y}^k=\mathbf{prox}_{\gamma f}(2\hat{x}^k-\hat{z}^k), \tag{19b}$$

$$z^{k+1}=z^k+2\eta_k U_{i_k}(\hat{y}^k-\hat{x}^k), \tag{19c}$$

where $\hat{x}^k$ and $\hat{y}^k$ are intermediate variables. Note that the order how the operators on f and g are applied affects both the sequence z^k [56] and whether coordinate-wise updates can be efficiently computed. Next, we present two special cases of (17) in Subsections 2.5.1 and 2.6 and discuss how to efficiently implement the update (19).

2.5.1 Feasibility problem

Suppose that $C_1,\dots,C_m$ are closed convex subsets of $\mathcal{H}$ with a nonempty intersection. The problem is to find a point in the intersection. Let $\mathcal{I}_{C_i}$ be the indicate function of the set C_i, that is, $\mathcal{I}_{C_i}(x)=0$ if $x\in C_i$ and ∞ otherwise. The feasibility problem can be formulated as the following convex minimization problem

$$\underset{x=(x_1,\dots,x_m)\in\mathcal{H}^m}{\text{minimize}}\ \sum_{i=1}^m\mathcal{I}_{C_i}(x_i)+\mathcal{I}_{\{x_1=\cdots=x_m\}}(x).$$

Let $z^k = (z_1^k, \ldots, z_m^k) \in \mathcal{H}^m$, $\hat{z}^k = (\hat{z}_1^k, \ldots, \hat{z}_m^k) \in \mathcal{H}^m$, and $\hat{\bar{z}}^k \in \mathcal{H}$. We can implement (19) as follows (see Appendix A for the derivation):

$$\hat{\bar{z}}^k = \tfrac{1}{m}\textstyle\sum_{i=1}^m \hat{z}_i^k \tag{20a}$$

$$\hat{y}_{i_k}^k = \mathrm{Proj}_{C_{i_k}}\big(2\hat{\bar{z}}^k - \hat{z}_{i_k}^k\big) \tag{20b}$$

$$z_{i_k}^{k+1} = z_{i_k}^k + 2\eta_k(\hat{y}_{i_k}^k - \hat{\bar{z}}^k). \tag{20c}$$

The update (20) can be efficiently implemented as follows. Let global memory hold $z_1, \ldots, z_m$ and maintain $\bar{z} = \frac{1}{m}\sum_{i=1}^m z_i$. At the kth update, an agent independently generates a random number $i_k \in \{1, \ldots, m\}$, then reads z_{i_k} as $\hat{z}_{i_k}^k$ and $\bar{z}$ as $\hat{\bar{z}}^k$, and finally computes $\hat{y}_{i_k}$ and updates z_{i_k} in global memory according to (20). Since $\bar{z}$ is maintained in global memory, the agent updates $\bar{z}$ according to $\bar{z}^{k+1} = \bar{z}^k + \frac{1}{m}(z_{i_k}^{k+1} - z_{i_k}^k)$. This implementation saves each agent from computing (20a) or reading $z_1, \ldots, z_m$. Each agent only reads z_{i_k} and $\bar{z}$, executes (20b), and updates z_{i_k} (by (20c)) and $\bar{z}$.

2.6 Async-parallel ADMM

Async-parallel ADMM is another application of (19). Consider the following problem

$$\underset{x\in\mathcal{H}_1, y\in\mathcal{H}_2}{\text{minimize}} \; f(x) + g(y) \quad \text{subject to } Ax + By = b, \tag{21}$$

where $\mathcal{H}_1$ and $\mathcal{H}_2$ are Hilbert spaces, A and B are bounded linear operators. We apply ARock along with update (18) or (19) to the Lagrange dual of (21):

$$\underset{w\in\mathcal{G}}{\text{minimize}} \; d_f(w) + d_g(w), \tag{22}$$

where $d_f(w) := f^*(A^*w)$, $d_g(w) := g^*(B^*w) - \langle w, b\rangle$, and f^* and g^* denote the convex conjugates of f and g, respectively [28]. We can compute the proximal maps induced by d_f and d_g by solving subproblems involving the terms in problem (21). It is well known that the proximal operator $z^+ = \mathbf{prox}_{\gamma\cdot d_f}(z)$ can be computed by (see Appendix A for the derivation)

$$\begin{cases} x^+ \in \arg\min_x f(x) - \langle z, Ax\rangle + \frac{\gamma}{2}\|Ax\|^2 \\ z^+ = z - \gamma A x^+, \end{cases} \tag{23}$$

and $z^+ = \mathbf{prox}_{\gamma\cdot d_g}(z)$ by

$$\begin{cases} y^+ \in \arg\min_y g(y) - \langle z, By - b\rangle + \frac{\gamma}{2}\|By - b\|^2 \\ z^+ = z - \gamma(By^+ - b). \end{cases} \tag{24}$$

Plugging (23) and (24) into (19), we obtain the following naive implementation

$$\hat{y}^k \in \arg\min_y g(y) - \langle \hat{z}^k, By - b\rangle + \frac{\gamma}{2}\|By - b\|^2 \tag{25a}$$

$$\hat{w}_g^k = \hat{z}^k - \gamma(B\hat{y}^k - b) \tag{25b}$$

$$\hat{x}^k \in \arg\min_x f(x) - \langle 2\hat{w}_g^k - \hat{z}^k, Ax\rangle + \frac{\gamma}{2}\|Ax\|^2 \tag{25c}$$

$$\hat{w}_f^k = 2\hat{w}_g^k - \hat{z}^k - \gamma A\hat{x}^k \tag{25d}$$

$$z_{i_k}^{k+1} = z_{i_k}^k + \eta_k(\hat{w}_{f,i_k}^k - \hat{w}_{g,i_k}^k). \tag{25e}$$

Under favorable structures, (25) can be implemented efficiently. For instance, when A and B are block diagonal matrices and f, g are corresponding block separable functions, steps (25a)–(25d) reduce to independent computation for each i. Since only $\hat{w}_{f,i_k}^k$ and $\hat{w}_{g,i_k}^k$ are needed to update the main variable z^k, we only need to compute (25a)–(25d) for the i_kth block. This is exploited in distributed and decentralized ADMM in the next two subsections.

2.6.1 Async-parallel ADMM for consensus optimization

Consider the following consensus optimization problem:

$$\underset{x_i, y\in\mathcal{H}}{\text{minimize}} \ \ \textstyle\sum_{i=1}^m f_i(x_i) \quad \text{subject to } x_i - y = 0, \quad \forall i = 1, ..., m \tag{26}$$

where $f_i(x_i)$ are proper close convex, possibly nonsmooth, functions. Rewrite (26) to the ADMM form:

$$\begin{aligned} &\underset{x_i, y\in\mathcal{H}}{\text{minimize}} \ \ \textstyle\sum_{i=1}^m f_i(x_i) + g(y) \\ &\text{subject to } \begin{bmatrix} I_{\mathcal{H}} & 0 & \cdots & 0 \\ 0 & I_{\mathcal{H}} & \cdots & 0 \\ & & \ddots & \\ 0 & 0 & \cdots & I_{\mathcal{H}} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} - \begin{bmatrix} I_{\mathcal{H}} \\ I_{\mathcal{H}} \\ \vdots \\ I_{\mathcal{H}} \end{bmatrix} y = 0, \end{aligned} \tag{27}$$

where $g = 0$. Now we can apply the async-parallel ADMM (25) to (27) with dual variables $z_1, ..., z_m \in \mathcal{H}$. The update (25a) reduces to

$$\hat{y}^k = \arg\min_y \left\{ \textstyle\sum_{i=1}^m \langle \hat{z}_i^k, y\rangle + \frac{\gamma m}{2}\|y\|^2 \right\} = (-\tfrac{1}{\gamma m}) \textstyle\sum_{i=1}^m \hat{z}_i^k.$$

Then, we can update (25b) as

$$(\hat{w}_{d_g}^k)_i = \hat{z}_i^k + \gamma\, \hat{y}^k, \tag{28}$$

Next, (25c) reduces to

$$\hat{x}_i^k = \arg\min_{x_i} \left\{ f_i(x_i) - \langle 2(\hat{w}_{d_g}^k)_i - \hat{z}_i^k, x_i\rangle + \tfrac{\gamma}{2}\|x_i\|^2 \right\}, \tag{29}$$

and (25d) to

$$(\hat{w}_{d_f}^k)_i = 2(\hat{w}_{d_g}^k)_i - \hat{z}_i^k - \gamma\, \hat{x}_i^k. \tag{30}$$

The updates (28)–(30) are only computed for the selected $i = i_k$ and then used in (25e). Therefore, we arrive at the following async-parallel ADMM algorithm for the consensus optimization problem (26).

Algorithm 4: ARock for consensus optimization

Input : sets shared variables $y^0, z_i^0,\ \forall i$, and $K > 0$

while $k < K$ *every agent asynchronously and continuously* **do**

choose i_k from $\{1, ..., m\}$ with equal probability;

evaluate $(\hat{w}_{d_g}^k)_{i_k}$, $\hat{x}_{i_k}^k$, and $(\hat{w}_{d_f}^k)_{i_k}$ following (28)–(30), respectively;

update $z_{i_k}^{k+1} = z_{i_k}^k + \eta_k\left((\hat{w}_{d_f}^k)_{i_k} - (\hat{w}_{d_g}^k)_{i_k}\right)$;

update $\hat{y}^{k+1} = \hat{y}^k + \frac{1}{\gamma m}(\hat{z}_{i_k}^k - \hat{z}_{i_k}^{k+1})$;

update the global counter $k \leftarrow k + 1$;

This algorithm applies to all of the distributed applications in the survey paper by Boyd et al.. [12]

2.6.2 Async-parallel ADMM for decentralized optimization

Let $V = \{1, ..., m\}$ be a set of agents and $E = \{(i, j) \mid \text{if agent } i \text{ connects to agent } j, i < j\}$ be a set of undirected edges between the agents. Let $G = (V, E)$. Consider the following decentralized consensus optimization problem on the graph G:

$$\begin{aligned} &\underset{x_1,...,x_m \in \mathbb{R}^d}{\text{minimize}} \ \ f(\mathbf{x}) := \textstyle\sum_{i=1}^m f_i(x_i) \\ &\text{subject to } x_i = x_j,\ \forall (i, j) \in E, \end{aligned} \tag{31}$$

where $x_1, ..., x_m \in \mathbb{R}^d$ are the local variables and each agent can only communicate with its neighbors in G. By introducing the auxiliary variable y_{ij} associated with each edge $(i, j) \in E$, the problem (31) can be reformulated as:

$$\begin{aligned} &\underset{x_i, y_{ij}}{\text{minimize}} \ \ \textstyle\sum_{i=1}^m f_i(x_i) \\ &\text{subject to } x_i = y_{ij},\ x_j = y_{ij}, \quad \forall (i, j) \in E. \end{aligned} \tag{32}$$

Define $x = (x_1, ..., x_m)^T$ and $y = (y_{ij})_{(i,j)\in E} \in \mathbb{R}^{|E|d}$ to rewrite (32) in a matrix form as

$$\begin{aligned} &\underset{x,y}{\text{minimize}} \ \ \textstyle\sum_{i=1}^m f_i(x_i) \\ &\text{subject to } Ax + By = 0, \end{aligned} \tag{33}$$

for proper matrices A and B. Applying the async-parallel ADMM (25) to (33) gives rise to the following simplified update: Let $E(i)$ be the set of edges connected with agent i and $|E(i)|$ be its cardinality. Let $L(i) = \{j \mid (j, i) \in E(i), j < i\}$ and $R(i) = \{j \mid (i, j) \in E(i), j > i\}$. To every pair of constraints $x_i = y_{ij}$ and $x_j = y_{ij}$, $(i, j) \in E$, we associate the dual variables $z_{ij,i}$ and $z_{ij,j}$, respectively. Whenever some agent i is activated, it calculates

$$\hat{x}_i^k = \arg\min_{x_i} f_i(x_i) + \Big(\sum_{l\in L(i)} \hat{z}_{li,l}^k + \sum_{r\in R(i)} \hat{z}_{ir,r}^k\Big)x_i + \frac{\gamma}{2}|E(i)| \cdot \|x_i\|^2, \tag{34a}$$

$$z_{ir,i}^{k+1} = z_{ir,i}^k - \eta_k((\hat{z}_{ir,i}^k + \hat{z}_{ir,r})/2 + \gamma\hat{x}_i^k), \quad \forall r \in R(i), \tag{34b}$$

$$z_{li,i}^{k+1} = z_{li,i}^k - \eta_k((\hat{z}_{li,i}^k + \hat{z}_{li,l})/2 + \gamma\hat{x}_i^k), \quad \forall l \in L(i). \tag{34c}$$

We present the algorithm based on (34) for problem (31) in Algorithm 5.

Algorithm 5: ARock for the decentralized problem (32)

Input : Each agent i sets the dual variables $z_{e,i}^0 = 0$ for $e \in E(i)$, $K > 0$.

while $k < K$, *any activated agent* i **do**

(previously received $\hat{z}_{li,l}^k$ from neighbors $l \in L(i)$ and $\hat{z}_{ir,r}^k$ from neighbors $r \in R(i)$);

update $\hat{x}_i^k$ according to (34a);

update $z_{li,i}^{k+1}$ and $z_{ir,i}^{k+1}$ according to (34b) and (34c), respectively;

send $z_{li,i}^{k+1}$ to neighbors $l \in L(i)$ and $z_{ir,i}^{k+1}$ to neighbors $r \in R(i)$.

Algorithm 5 activates one agent at each iteration and updates all the dual variables associated with the agent. In this case, only one-sided communication is needed, for sending the updated dual variables in the last step. We allow this communication to be delayed in the sense that some neighbor agents of the agent i may be activated and start their computation before receiving the latest dual variables from the agent i.

Our algorithm is different from the asynchronous ADMM algorithm by Wei and Ozdaglar [53]. Their algorithm activates an edge and its two associated agents at each iteration and thus requires two-sided communication at each activation. We can recover their algorithm [53] as a special case of ARock by activating an edge $(i, j) \in E$ and its associated agents i and j at each iteration, updating the two dual variables $z_{ij,i}$ and $z_{ij,j}$ associated with the edge, as well as computing the intermediate variables x_i, x_j, and y_{ij}. The updates are derived from (33) with the order of x and y swapped. Note that [53] does not consider the case that adjacent edges are both activated in a short period of time and cause their computing to be overlapped and communication to be delayed. Indeed, their algorithm corresponds to $\tau = 0$ and the corresponding stepsize $\eta_k \equiv 1$. Appendix B presents the steps to derive the algorithms in this subsection.

3 Convergence

This section provides convergence analysis of ARock (Algorithm 1). We show a.s. weak convergence for a nonexpansive operator T and a.s. strong convergence for nonexpansive and demicompact T in Subsection 3.1, and then linear convergence for a quasi-strongly monotone operator $S \equiv I - T$ in Subsection 3.2.

3.1 Almost sure convergence

Throughout the following analysis, we let $p_{\min} = \min_i p_i > 0$ and $|J(k)|$ be the number of elements in $J(k)$ (see Subsection 1.2). Define

$$\bar{x}^{k+1} := x^k - \eta_k S\hat{x}^k. \tag{35}$$

By Lemma 1 below, T is nonexpansive if and only if S is 1/2-cocoercive.

Lemma 1 *Operator $T : \mathcal{H} \to \mathcal{H}$ is nonexpansive if and only if $S = I - T$ is 1/2-cocoercive, i.e.,*

$$\langle x - y, Sx - Sy\rangle \geq \frac{1}{2}\|Sx - Sy\|^2, \quad \forall\, x, y \in \mathcal{H}. \tag{36}$$

Proof See textbook [6, Proposition 4.33] for the proof of the "if" part, and the "only if" part is just the reverse of the proof.

The following lemma shows that the conditional expectation of the distance between x^{k+1} and any $x^* \in \operatorname{Fix} T$ for given $\mathcal{X}^k = \{x^0, x^1, \cdots, x^k\}$ has an upper bound that depends on $\mathcal{X}^k$ and x^* only.

Lemma 2 *Let $(x^k)_{k\geq 0}$ be the sequence generated by Algorithm 1. Then for any $x^* \in \operatorname{Fix} T$, we have*

$$\begin{aligned}\mathbb{E}\big(\|x^{k+1}-x^*\|^2 \,|\, \mathcal{X}^k\big) \leq & \|x^k - x^*\|^2 + \frac{\gamma}{m}\sum_{d\in J(k)}\|x^d - x^{d+1}\|^2 \\ & + \frac{1}{m}\left(\frac{|J(k)|}{\gamma} + \frac{1}{mp_{\min}} - \frac{1}{\eta_k}\right)\|x^k - \bar{x}^{k+1}\|^2\end{aligned} \tag{37}$$

where $\gamma > 0$ (to be optimized later) and $\mathbb{E}(\cdot \,|\, \mathcal{X}^k)$ denotes expectation conditional on $\mathcal{X}^k$.

Proof We have

$$\begin{aligned}& \mathbb{E}\left(\|x^{k+1}-x^*\|^2 \,|\, \mathcal{X}^k\right) \\ \overset{(3)}{=} & \mathbb{E}\left(\|x^k - \tfrac{\eta_k}{mp_{i_k}}S_{i_k}\hat{x}^k - x^*\|^2 \,|\, \mathcal{X}^k\right) \\ = & \|x^k - x^*\|^2 + \mathbb{E}\left(\tfrac{2\eta_k}{mp_{i_k}}\left\langle S_{i_k}\hat{x}^k, x^* - x^k\right\rangle + \tfrac{\eta_k^2}{m^2p_{i_k}^2}\|S_{i_k}\hat{x}^k\|^2 \,\Big|\, \mathcal{X}^k\right) \\ = & \|x^k - x^*\|^2 + \tfrac{2\eta_k}{m}\textstyle\sum_{i=1}^m\left\langle S_i\hat{x}^k, x^* - x^k\right\rangle + \tfrac{\eta_k^2}{m^2}\sum_{i=1}^m\tfrac{1}{p_i}\|S_i\hat{x}^k\|^2 \\ = & \|x^k - x^*\|^2 + \tfrac{2\eta_k}{m}\left\langle S\hat{x}^k, x^* - x^k\right\rangle + \tfrac{\eta_k^2}{m^2}\textstyle\sum_{i=1}^m\tfrac{1}{p_i}\|S_i\hat{x}^k\|^2,\end{aligned} \tag{38}$$

where the third equality holds because the probability of choosing i is p_i.

Note that

$$\begin{aligned}\textstyle\sum_{i=1}^m\tfrac{1}{p_i}\|S_i\hat{x}^k\|^2 &\leq \frac{1}{p_{\min}}\sum_{i=1}^m\|S_i\hat{x}^k\|^2 \\ &= \frac{1}{p_{\min}}\|S\hat{x}^k\|^2 \overset{(35)}{=} \tfrac{1}{\eta_k^2 p_{\min}}\|x^k - \bar{x}^{k+1}\|^2,\end{aligned} \tag{39}$$

and

$$\begin{aligned}& \left\langle S\hat{x}^k, x^* - x^k\right\rangle \\ \overset{(5)}{=} & \left\langle S\hat{x}^k, x^* - \hat{x}^k + \textstyle\sum_{d\in J(k)}(x^d - x^{d+1})\right\rangle \\ \overset{(35)}{=} & \left\langle S\hat{x}^k, x^* - \hat{x}^k\right\rangle + \tfrac{1}{\eta_k}\textstyle\sum_{d\in J(k)}\langle x^k - \bar{x}^{k+1}, x^d - x^{d+1}\rangle \\ \leq & \left\langle S\hat{x}^k - Sx^*, x^* - \hat{x}^k\right\rangle + \tfrac{1}{2\eta_k}\textstyle\sum_{d\in J(k)}\left(\tfrac{1}{\gamma}\|x^k - \bar{x}^{k+1}\|^2 + \gamma\|x^d - x^{d+1}\|^2\right) \\ \overset{(36)}{\leq} & -\tfrac{1}{2}\|S\hat{x}^k\|^2 + \tfrac{1}{2\eta_k}\textstyle\sum_{d\in J(k)}(\tfrac{1}{\gamma}\|x^k - \bar{x}^{k+1}\|^2 + \gamma\|x^d - x^{d+1}\|^2) \\ \overset{(35)}{=} & -\tfrac{1}{2\eta_k^2}\|x^k - \bar{x}^{k+1}\|^2 + \tfrac{|J(k)|}{2\gamma\eta_k}\|x^k - \bar{x}^{k+1}\|^2 + \tfrac{\gamma}{2\eta_k}\textstyle\sum_{d\in J(k)}\|x^d - x^{d+1}\|^2,\end{aligned} \tag{40}$$

where the first inequality follows from the Young's inequality. Plugging (39) and (40) into (38) gives the desired result.

We will introduce a special metric, under which the sequence $(x^k)_{k\geq 0}$ generated by Algorithm 1 follows a non-negative almost supermartingale and its convergence result as follows:

Lemma 3 ([47, Theorem 1]) *Let $\mathscr{F} = (\mathcal{F}^k)_{k\geq 0}$ be a sequence of sub-sigma algebras of $\mathcal{F}$ such that $\forall k \geq 0$, $\mathcal{F}^k \subset \mathcal{F}^{k+1}$. Define $\ell_+(\mathscr{F})$ as the set of sequences of $[0,+\infty)$-valued random variables $(\xi_k)_{k\geq 0}$, where ξ_k is $\mathcal{F}^k$ measurable, and $\ell_+^1(\mathscr{F}) := \{(\xi_k)_{k\geq 0} \in \ell_+(\mathscr{F}) | \sum_k \xi_k < +\infty \text{ a.s.}\}$. Let $(\alpha_k)_{k\geq 0}, (v_k)_{k\geq 0} \in \ell_+(\mathscr{F})$, and $(\eta_k)_{k\geq 0}, (\xi_k)_{k\geq 0} \in \ell_+^1(\mathscr{F})$ be such that*

$$\mathbb{E}(\alpha_{k+1}|\mathcal{F}^k) + v_k \leq (1+\xi_k)\alpha_k + \eta_k.$$

Then $(v_k)_{k\geq 0} \in \ell_+^1(\mathscr{F})$, and α_k converges to a $[0,+\infty)$-valued random variable a.s..

Let $\mathcal{H}^{\tau+1} = \prod_{i=0}^{\tau} \mathcal{H}$ be a product space and $\langle \cdot \,|\, \cdot \rangle$ be the induced inner product:

$$\langle (z^0,\ldots,z^\tau) \,|\, (y^0,\ldots,y^\tau)\rangle = \sum_{i=0}^{\tau} \langle z^i, y^i\rangle, \quad \forall (z^0,\ldots,z^\tau), (y^0,\ldots,y^\tau) \in \mathcal{H}^{\tau+1}.$$

Define a $(\tau+1)\times(\tau+1)$ matrix M' by

$$M' := \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} + \sqrt{p_{\min}} \begin{bmatrix} \tau & -\tau & & & & \\ -\tau & 2\tau-1 & 1-\tau & & & \\ & 1-\tau & 2\tau-3 & 2-\tau & & \\ & & \ddots & \ddots & \ddots & \\ & & & -2 & 3 & -1 \\ & & & & -1 & 1 \end{bmatrix},$$

and let $M = M' \otimes I_{\mathcal{H}}$. Here $I_{\mathcal{H}}$ is the identity operator on $\mathcal{H}$, and $\otimes$ represents the Kronecker product. For a given $(y^0,\cdots,y^\tau) \in \mathcal{H}^{\tau+1}$, $(z^0,\cdots,z^\tau) = M(y^0,\cdots,y^\tau)$ is given by:

$$\begin{aligned} z^0 &= y^0 + \sqrt{p_{\min}}(y^0 - y^1), \\ z^i &= \sqrt{p_{\min}}((i-\tau-1)y^{i-1} + (2\tau-2i+1)y^i + (i-\tau)y^{i+1}), \text{ if } 1 \leq i \leq \tau-1, \\ z^\tau &= \sqrt{p_{\min}}(y^\tau - y^{\tau-1}). \end{aligned}$$

Then M is a self-adjoint and positive definite linear operator since M' is symmetric and positive definite, and we define $\langle \cdot \,|\, \cdot \rangle_M = \langle \cdot \,|\, M\cdot \rangle$ as the M-weighted inner product and $\|\cdot\|_M$ the induced norm.

Let

$$\mathbf{x}^k = (x^k, x^{k-1}, \ldots, x^{k-\tau}) \in \mathcal{H}^{\tau+1}, \ k \geq 0, \text{ and } \mathbf{x}^* = (x^*, x^*, \ldots, x^*) \in \mathbf{X}^* \subseteq \mathcal{H}^{\tau+1},$$

where we set $x^k = x^0$ for $k < 0$. With

$$\xi_k(\mathbf{x}^*) := \|\mathbf{x}^k - \mathbf{x}^*\|_M^2 = \|x^k - x^*\|^2 + \sqrt{p_{\min}} \textstyle\sum_{i=k-\tau}^{k-1} (i-(k-\tau)+1)\|x^i - x^{i+1}\|^2, \quad \forall\, k \geq 0, \tag{41}$$

we have the following fundamental inequality:

Theorem 2 (Fundamental inequality) *Let $(x^k)_{k\geq 0}$ be the sequence generated by ARock. Then for any $\mathbf{x}^* \in \mathbf{X}^*$, it holds that*

$$\mathbb{E}\left(\xi_{k+1}(\mathbf{x}^*) \,\middle|\, \mathcal{X}^k\right) \leq \xi_k(\mathbf{x}^*) + \frac{1}{m}\left(\frac{2\tau}{m\sqrt{p_{\min}}} + \frac{1}{mp_{\min}} - \frac{1}{\eta_k}\right)\|\bar{x}^{k+1} - x^k\|^2. \tag{42}$$

Proof Let $\gamma = m\sqrt{p_{\min}}$. We have

$$\begin{aligned}
&\mathbb{E}(\xi_{k+1}(\mathbf{x}^*)|\mathcal{X}^k)\\
\overset{(41)}{=}&\mathbb{E}(\|x^{k+1}-x^*\|^2|\mathcal{X}^k) + \gamma\textstyle\sum_{i=k+1-\tau}^{k}\frac{i-(k-\tau)}{m}\mathbb{E}(\|x^i-x^{i+1}\|^2|\mathcal{X}^k)\\
\overset{(3)}{=}&\mathbb{E}(\|x^{k+1}-x^*\|^2|\mathcal{X}^k) + \tfrac{\gamma\tau}{m}\mathbb{E}(\tfrac{\eta_k^2}{m^2p_{i_k}^2}\|S_{i_k}\hat{x}^k\|^2|\mathcal{X}^k) + \gamma\textstyle\sum_{i=k+1-\tau}^{k-1}\frac{i-(k-\tau)}{m}\|x^i-x^{i+1}\|^2\\
\leq&\mathbb{E}(\|x^{k+1}-x^*\|^2|\mathcal{X}^k) + \tfrac{\gamma\tau}{m^3p_{\min}}\|x^k-\bar{x}^{k+1}\|^2 + \gamma\textstyle\sum_{i=k+1-\tau}^{k-1}\frac{i-(k-\tau)}{m}\|x^i-x^{i+1}\|^2\\
\overset{(37)}{\leq}&\|x^k-x^*\|^2 + \tfrac{1}{m}\left(\tfrac{|J(k)|}{\gamma} + \tfrac{\gamma\tau}{m^2p_{\min}} + \tfrac{1}{mp_{\min}} - \tfrac{1}{\eta_k}\right)\|x^k-\bar{x}^{k+1}\|^2\\
&+\tfrac{\gamma}{m}\textstyle\sum_{d\in J(k)}\|x^d-x^{d+1}\|^2 + \gamma\sum_{i=k+1-\tau}^{k-1}\frac{i-(k-\tau)}{m}\|x^i-x^{i+1}\|^2\\
\leq&\|x^k-x^*\|^2 + \tfrac{1}{m}\left(\tfrac{\tau}{\gamma} + \tfrac{\gamma\tau}{m^2p_{\min}} + \tfrac{1}{mp_{\min}} - \tfrac{1}{\eta_k}\right)\|x^k-\bar{x}^{k+1}\|^2\\
&+\tfrac{\gamma}{m}\textstyle\sum_{i=k-\tau}^{k-1}\|x^i-x^{i+1}\|^2 + \gamma\sum_{i=k+1-\tau}^{k-1}\frac{i-(k-\tau)}{m}\|x^i-x^{i+1}\|^2\\
\overset{(41)}{=}&\xi_k(\mathbf{x}^*) + \tfrac{1}{m}\left(\tfrac{2\tau}{m\sqrt{p_{\min}}} + \tfrac{1}{mp_{\min}} - \tfrac{1}{\eta_k}\right)\|x^k-\bar{x}^{k+1}\|^2.
\end{aligned}$$

The first inequality follows from the computation of the expectation and (39), the second inequality holds because $J(k)\subset\{k-1,k-2,\cdots,k-\tau\}$, and the last equality uses $\gamma = m\sqrt{p_{\min}}$, which minimizes $\frac{\tau}{\gamma}+\frac{\gamma\tau}{m^2p_{\min}}$ over $\gamma>0$. Hence, the desired inequality holds.

Remark 1 (Stochastic Fejér monotonicity) From (42), if $0<\eta_k\leq\frac{mp_{\min}}{2\tau\sqrt{p_{\min}}+1}$, then we have $\mathbb{E}(\|\mathbf{x}^{k+1}-\mathbf{x}^*\|_M^2|\mathcal{X}^k)\leq\|\mathbf{x}^k-\mathbf{x}^*\|_M^2$, $\forall\mathbf{x}^*\in\mathbf{X}^*$, namely, $(\mathbf{x}^k)_{k\geq0}$ is stochastic Fejér monotone.

The next lemma is a direct consequence of the invertibility of M.

Lemma 4 *A sequence $(\mathbf{z}^k)_{k\geq0}\subset\mathcal{H}^{\tau+1}$ (weakly) converges to $\mathbf{z}\in\mathcal{H}^{\tau+1}$ under the metric $\langle\cdot\,|\,\cdot\rangle$ if and only if it does so under the metric $\langle\cdot\,|\,\cdot\rangle_M$.*

In light of Lemma 4, the next lemma does not specify the inner product used for weak convergence. Most parts of the lemma directly follow from the results in [17].

Lemma 5 *Let $(x^k)_{k\geq0}\subset\mathcal{H}$ be the sequence generated by ARock with $\eta_k\in[\eta_{\min},\frac{cmp_{\min}}{2\tau\sqrt{p_{\min}}+1}]$ for certain $\eta_{\min}>0$ and $0<c<1$. Then the following statements hold:*

(i) $\sum_{k=0}^{\infty}\|x^k-\bar{x}^{k+1}\|^2<\infty$ *a.s.*.
(ii) $x^k-x^{k+1}\to0$ *a.s. and* $\hat{x}^k-x^{k+1}\to0$ *a.s.;*
(iii) *The sequence $(\mathbf{x}^k)_{k\geq0}\subset\mathcal{H}^{\tau+1}$ is bounded a.s.;*
(iv) *There exists $\tilde{\Omega}\in\mathcal{F}$ such that $P(\tilde{\Omega})=1$ and, for every $\omega\in\tilde{\Omega}$ and every $\mathbf{x}^*\in\mathbf{X}^*$, $(\|\mathbf{x}^k(\omega)-\mathbf{x}^*\|_M)_{k\geq0}$ converges.*
(v) *Let $\mathscr{Z}(\mathbf{x}^k)$ be the set of weakly convergent cluster points of $(\mathbf{x}^k)_{k\geq0}$. Then, $\mathscr{Z}(\mathbf{x}^k)\subseteq\mathbf{X}^*$ a.s..*

Proof (i): Applying Lemma 3 to (42) gives this result directly.

(ii) From (i), we have $x^k-\bar{x}^{k+1}\to0$ a.s.. Since $\|x^k-x^{k+1}\|\leq\frac{1}{mp_{\min}}\|x^k-\bar{x}^{k+1}\|$, we have $x^k-x^{k+1}\to0$ a.s.. Then from (5), we have $\hat{x}^k-x^k\to0$ a.s..

(iii): From Lemma 3, we have that $(\|\mathbf{x}^k-\mathbf{x}^*\|_M^2)_{k\geq 0}$ converges a.s., and so does $(\|\mathbf{x}^k-\mathbf{x}^*\|_M)_{k\geq 0}$. Hence, we have $\lim_{k\to\infty}\|\mathbf{x}^k-\mathbf{x}^*\|_M=\gamma$ a.s., where γ is a $[0,+\infty)$-valued random variable. Hence, $(\|\mathbf{x}^k-\mathbf{x}^*\|_M)_{k\geq 0}$ must be bounded a.s., and so is $(\mathbf{x}^k)_{k\geq 0}$.

(iv): The proof follows directly from [17, Proposition 2.3 (iii)]. It is worth noting that $\tilde{\Omega}$ in the statement works for all $\mathbf{x}^*\in\mathbf{X}^*$, namely, Ω does not depend on $\mathbf{x}^*$.

(v): By (ii), there exists $\hat{\Omega}\in\mathcal{F}$ such that $P(\hat{\Omega})=1$ and

$$x^k(w)-x^{k+1}(w)\to 0,\quad \forall w\in\hat{\Omega}. \tag{43}$$

For any $\omega\in\hat{\Omega}$, let $(\mathbf{x}^{k_n}(\omega))_{n\geq 0}$ be a weakly convergent subsequence of $(\mathbf{x}^k(\omega))_{k\geq 0}$, i.e., $\mathbf{x}^{k_n}(\omega)\rightharpoonup\mathbf{x}$, where $\mathbf{x}^{k_n}(\omega)=(x^{k_n}(\omega),x^{k_n-1}(\omega)...,x^{k_n-\tau}(\omega))$ and $\mathbf{x}=(u^0,...,u^\tau)$. Note that $\mathbf{x}^{k_n}(\omega)\rightharpoonup\mathbf{x}$ implies $x^{k_n-j}(\omega)\rightharpoonup u^j$, $\forall j$. Therefore, $u^i=u^j$, for any $i,j\in\{0,\cdots,\tau\}$ because $x^{k_n-i}(\omega)-x^{k_n-j}(\omega)\to 0$.

Furthermore, observing $\eta_k\geq\eta_{\min}>0$, we have

$$\begin{aligned}\lim_{n\to\infty}\hat{x}^{k_n}(\omega)-T\hat{x}^{k_n}(\omega)&=\lim_{n\to\infty}S\hat{x}^{k_n}(\omega)\\&=\lim_{n\to\infty}\frac{1}{\eta_{k_n}}(x^{k_n}(\omega)-\bar{x}^{k_n+1}(\omega))=0.\end{aligned} \tag{44}$$

From the triangle inequality and the nonexpansiveness of T, it follows that

$$\begin{aligned}&\left\|x^{k_n}(\omega)-Tx^{k_n}(\omega)\right\|\\=&\left\|x^{k_n}(\omega)-\hat{x}^{k_n}(\omega)+\hat{x}^{k_n}(\omega)-T\hat{x}^{k_n}(\omega)+T\hat{x}^{k_n}(\omega)-Tx^{k_n}(\omega)\right\|\\\leq&\left\|x^{k_n}(\omega)-\hat{x}^{k_n}(\omega)\right\|+\left\|\hat{x}^{k_n}(\omega)-T\hat{x}^{k_n}(\omega)\right\|+\left\|T\hat{x}^{k_n}(\omega)-Tx^{k_n}(\omega)\right\|\\\leq&2\left\|x^{k_n}(\omega)-\hat{x}^{k_n}(\omega)\right\|+\|\hat{x}^{k_n}(\omega)-T\hat{x}^{k_n}(\omega)\|\\\leq&2\textstyle\sum_{d\in J(k_n)}\left\|x^d(\omega)-x^{d+1}(\omega)\right\|+\|\hat{x}^{k_n}(\omega)-T\hat{x}^{k_n}(\omega)\|\end{aligned}$$

By (43) and (44), we have from the above inequality that $\lim_{n\to\infty}x^{k_n}(\omega)-Tx^{k_n}(\omega)=0$. Now the demiclosedness principle [6, Theorem 4.17] implies $u^0\in\operatorname{Fix}T$, and this completes the proof of (v).

Theorem 3 *Under the same assumption as in Lemma 5, the sequence $(\mathbf{x}^k)_{k\geq 0}$ weakly converges to an $\mathbf{X}^*$-valued random variable a.s.. In addition, if the operator T is demicompact at 0, $(\mathbf{x}^k)_{k\geq 0}$ strongly converges to an $\mathbf{X}^*$-valued random variable a.s..*

Proof The proof for a.s. weak convergence follows directly from [17, Proposition 2.3 (iv)].

Next we assume that T is demicompact at 0. From the proof of Lemma 5 (v), there exists $\hat{\Omega}\in\mathcal{F}$ such that $P(\hat{\Omega})=1$ and for any $w\in\hat{\Omega}$ and any weakly convergent subsequence of $(\mathbf{x}^{k_n}(w))_{n\geq 0}$, $\lim_{n\to\infty}x^{k_n}(w)-Tx^{k_n}(w)=0$. By the demicompactness of T, the sequence $(x^{k_n}(w))_{n\geq 0}$ has a strongly convergent subsequence, for which we still use $(x^{k_n}(w))_{n\geq 0}$. Hence, $x^{k_n}(w)\to\bar{x}(w)\in\operatorname{Fix}T$. From Lemma 5 (ii), we get $\mathbf{x}^{k_n}(w)\to\bar{\mathbf{x}}(w)\in\mathbf{X}^*$. Then by Lemma 5 (iv), there exists $\tilde{\Omega}\in\mathcal{F}$ such that $P(\tilde{\Omega})=1$ and for every $w\in\tilde{\Omega}$ and every $\mathbf{x}^*\in\mathbf{X}^*$, $(\|\mathbf{x}^k(w)-\mathbf{x}^*\|_M)_{k\geq 0}$ converges. Thus, for any $w\in\hat{\Omega}\cap\tilde{\Omega}$, we have $\lim_{k\to\infty}\|\mathbf{x}^k(w)-\bar{\mathbf{x}}(w)\|_M=0$. Because $P(\hat{\Omega}\cap\tilde{\Omega})=1$, we conclude that $(\mathbf{x}^k)_{k\geq 0}$ strongly converges to an $\mathbf{X}^*$-valued random variable a.s..

3.2 Linear convergence for quasi-strongly monotone operator S

In this section, we establish linear convergence for Algorithm 1 under the assumption that S is quasi-strongly monotone. We first establish three lemmas.

Lemma 6 *Let $(a_k)_{k\geq 0}$ be a sequence of positive scalars satisfying*

$$a_{k+1} \leq \rho \max_{k-\kappa\leq j\leq k} a_j, \tag{45}$$

for some constant $\rho \in (0,1)$ and non-negative integer κ. Then we have

$$a_k \leq a_0 \rho^{k/(\kappa+1)}.$$

Proof When $k=0$, we have $a_0 = a_0\rho^{0/(\kappa+1)}$. Next, we assume that $a_k \leq a_0^{k/(\kappa+1)}$ holds for all $k \leq K$. Then we have

$$a_{K+1} \leq \rho \max_{K-\kappa\leq j\leq K} a_j \leq \rho a_0 \rho^{(K-\kappa)/(\kappa+1)} = a_0\rho^{(K+1)/(\kappa+1)}.$$

By induction, we have $a_k \leq a_0\rho^{k/(\kappa+1)}$, $\forall k$, and complete the proof.

Lemma 7 *Let $(x^k)_{k\geq 0}$ be the sequence generated by ARock. With consistent read, we have*

$$\mathbb{E}\|\hat{x}^k - x^*\|^2 \leq \max_{k-\tau\leq j\leq k} \mathbb{E}\|x^j - x^*\|^2, \tag{46}$$

$$\mathbb{E}\|\hat{x}^k - x^k\|^2 \leq \frac{4\tau\sum_{d=k-\tau}^{k-1}\eta_d^2}{m^2 p_{\min}} \max_{k-2\tau\leq j\leq k-1} \mathbb{E}\|x^j - x^*\|^2, \tag{47}$$

and with inconsistent read, we have

$$\mathbb{E}\|\hat{x}^k - x^*\|^2 \leq 2\mathbb{E}\|x^k - x^*\|^2 + \left(\frac{8\tau}{m^2 p_{\min}}\sum_{d=k-\tau}^{k-1}\eta_d^2(\tau+1)^2\right) \max_{k-2\tau\leq j\leq k-1} \mathbb{E}\|x^j - x^*\|^2, \tag{48}$$

$$\mathbb{E}\|\hat{x}^k - x^k\|^2 \leq \frac{4\tau(\tau+1)^2\sum_{d=k-\tau}^{k-1}\eta_d^2}{m^2 p_{\min}} \max_{k-2\tau\leq j\leq k-1} \mathbb{E}\|x^j - x^*\|^2. \tag{49}$$

Proof By consistent read, we have $\hat{x}^k = x^{k-j_k}$ for some $0 \leq j_k \leq \tau$. Then, (46) immediately holds, and

$$\begin{aligned}
\mathbb{E}\|\hat{x}^k - x^k\|^2 \leq& |J(k)| \sum_{d\in J(k)} \mathbb{E}\|x^d - x^{d+1}\|^2 \\
\leq& \tau \sum_{d=k-\tau}^{k-1} \frac{\eta_d^2}{m^2 p_{\min}} \mathbb{E}\|S\hat{x}^d\|^2 \\
\overset{(36)}{\leq}& \frac{4\tau}{m^2 p_{\min}} \sum_{d=k-\tau}^{k-1} \eta_d^2 \mathbb{E}\|\hat{x}^d - x^*\|^2.
\end{aligned} \tag{50}$$

The first inequality follows from the Cauchy-Schwarz inequality, and the second inequality follows from (39). Hence, (47) is obtained by plugging (46) into (50). With inconsistent read, we have

$$\begin{aligned}
\mathbb{E}\|\hat{x}^k - x^*\|^2 =& \mathbb{E}\left\|x^k + \sum_{d\in J(k)}(x^d - x^{d+1}) - x^*\right\|^2 \\
\leq& (\tau+1)\sum_{d=k-\tau}^{k} \mathbb{E}\|x^d - x^*\|^2 \\
\leq& (\tau+1)^2 \max_{k-\tau\leq j\leq k} \mathbb{E}\|x^j - x^*\|^2,
\end{aligned} \tag{51}$$

and (49) follows by plugging (51) into (50). In addition,

$$
\begin{aligned}
&\mathbb{E}\|\hat{x}^k - x^*\|^2\\
=&\mathbb{E}\left\|x^k + \textstyle\sum_{d\in J(k)}(x^d - x^{d+1}) - x^*\right\|^2\\
\le&\mathbb{E}\|x^k - x^*\|^2 + 2\sum_{d\in J(k)}\mathbb{E}\langle x^k - x^*, x^d - x^{d+1}\rangle + |J(k)|\sum_{d\in J(k)}\mathbb{E}\|x^d - x^{d+1}\|^2\\
\le&\mathbb{E}\|x^k - x^*\|^2 + \sum_{d\in J(k)}\left(\frac{1}{\tau}\mathbb{E}\|x^k - x^*\|^2 + \tau\mathbb{E}\|x^d - x^{d+1}\|^2\right) + \tau\sum_{d\in J(k)}\mathbb{E}\|x^d - x^{d+1}\|^2\\
\le&2\mathbb{E}\|x^k - x^*\|^2 + 2\tau\sum_{d\in J(k)}\mathbb{E}\|x^d - x^{d+1}\|^2\\
\le&2\mathbb{E}\|x^k - x^*\|^2 + \frac{8\tau}{m^2 p_{\min}}\sum_{d=k-\tau}^{k-1}\eta_d^2\mathbb{E}\|\hat{x}^d - x^*\|^2\\
\overset{(51)}{\le}&2\mathbb{E}\|x^k - x^*\|^2 + \frac{8\tau}{m^2 p_{\min}}\sum_{d=k-\tau}^{k-1}\eta_d^2(\tau+1)^2\max_{d-\tau\le j\le d}\mathbb{E}\|x^j - x^*\|^2\\
\le&2\mathbb{E}\|x^k - x^*\|^2 + \left(\frac{8\tau}{m^2 p_{\min}}\sum_{d=k-\tau}^{k-1}\eta_d^2(\tau+1)^2\right)\max_{k-2\tau\le j\le k-1}\mathbb{E}\|x^j - x^*\|^2,
\end{aligned}
\tag{52}
$$

where the fourth inequality follows from the same arguments for (50). This completes the proof.

Lemma 8 *Assume that S is quasi-μ-strongly monotone with $\mu > 0$. Let $(x^k)_{k\ge 0}$ be the sequence generated by ARock. Then for any $x^* \in \operatorname{Fix} T$, we have*

$$
\mathbb{E}\|x^{k+1} - x^*\|^2 \le a_k\mathbb{E}\|x^k - x^*\|^2 + b_k\max_{k-2\tau\le j\le k}\mathbb{E}\|x^j - x^*\|^2 \tag{53}
$$

where

$$
a_k = \begin{cases} 1 - \frac{2\eta_k\mu}{m} + \frac{\eta_k\beta}{m}, & \textit{consistent read},\\ 1 - \frac{2\eta_k\mu}{m} + \frac{\eta_k\beta}{m} + \frac{8\eta_k^2}{m^2 p_{\min}}, & \textit{inconsistent read}, \end{cases}
$$

$$
b_k = \begin{cases} \frac{16\eta_k\tau\sum_{d=k-\tau}^{k-1}\eta_d^2}{m^3\beta p_{\min}} + \frac{4\eta_k^2}{m^2 p_{\min}}, & \textit{consistent read},\\ \frac{16\eta_k\tau(\tau+1)^2\sum_{d=k-\tau}^{k-1}\eta_d^2}{m^3 p_{\min}\beta} + \frac{32\eta_k^2\tau(\tau+1)^2\sum_{d=k-\tau}^{k-1}\eta_d^2}{m^4 p_{\min}^2}, & \textit{inconsistent read}, \end{cases}
$$

with any constant $\beta > 0$.

Proof For any $x^* \in \operatorname{Fix} T$, we start from (38) and get that

$$\begin{aligned}
&\mathbb{E}\left(\|x^{k+1}-x^*\|^2|\mathcal{X}^k\right)\\
=&\|x^k-x^*\|^2+\tfrac{2\eta_k}{m}\langle S\hat{x}^k, x^*-x^k\rangle+\tfrac{\eta_k^2}{m^2}\textstyle\sum_{i=1}^m \frac{1}{p_i}\|S_i\hat{x}^k\|^2\\
\le&\|x^k-x^*\|^2+\tfrac{2\eta_k}{m}\langle Sx^k, x^*-x^k\rangle+\tfrac{2\eta_k}{m}\langle S\hat{x}^k-Sx^k, x^*-x^k\rangle+\tfrac{\eta_k^2}{m^2p_{\min}}\|S\hat{x}^k\|^2\\
\le&(1-2\eta_k\mu/m)\|x^k-x^*\|^2+\tfrac{2\eta_k}{m}\langle S\hat{x}^k-Sx^k, x^*-x^k\rangle+\tfrac{\eta_k^2}{m^2p_{\min}}\|S\hat{x}^k\|^2\\
\le&(1-2\eta_k\mu/m+\eta_k\beta/m)\|x^k-x^*\|^2+\tfrac{\eta_k}{m\beta}\|S\hat{x}^k-Sx^k\|^2+\tfrac{\eta_k^2}{m^2p_{\min}}\|S\hat{x}^k\|^2\\
\le&(1-2\eta_k\mu/m+\eta_k\beta/m)\|x^k-x^*\|^2+\tfrac{4\eta_k}{m\beta}\|\hat{x}^k-x^k\|^2+\tfrac{4\eta_k^2}{m^2p_{\min}}\|\hat{x}^k-x^*\|^2,
\end{aligned}$$

where the first inequality follows from (39), the second inequality from the quasi-strong monotonicity of S, and the last inequality from (36). Taking expectation on both sides gives us

$$\mathbb{E}\|x^{k+1}-x^*\|^2 \le (1-2\eta_k\mu/m+\eta_k\beta/m)\mathbb{E}\|x^k-x^*\|^2+\tfrac{4\eta_k}{m\beta}\mathbb{E}\|\hat{x}^k-x^k\|^2+\tfrac{4\eta_k^2}{m^2p_{\min}}\mathbb{E}\|\hat{x}^k-x^*\|^2.$$

Now use Lemma 7 to get the desired result by bounding $\mathbb{E}\|\hat{x}^k-x^k\|^2$ and $\mathbb{E}\|\hat{x}^k-x^*\|^2$ in the above inequality. This completes the proof.

Now we are ready to derive the linear convergence rate of ARock.

Theorem 4 (Linear convergence for quasi-μ-strongly monotone operator) *Assume that S is quasi-μ-strongly monotone with $\mu>0$. Let $(x^k)_{k\ge 0}$ be the sequence generated by ARock with a constant stepsize $\eta \in (0, \frac{\mu m p_{\min}}{2(1+2\tau\sqrt{p_{\min}})})$ for consistent read, and $\eta \in (0, \frac{\mu m p_{\min}}{4\sqrt{p_{\min}}\tau(\tau+1)+4+\mu^2})$ for inconsistent read. Then*

$$\mathbb{E}\|x^k-x^*\|^2 \le \|x^0-x^*\|^2\rho^{k/(2\tau+1)}, \tag{54}$$

where $0<\rho<1$ is given by

$$\begin{cases}
\rho = 1-\frac{2\eta\mu}{m}+\frac{8\eta^2\tau}{m^2\sqrt{p_{\min}}}+\frac{4\eta^2}{m^2p_{\min}}, & \text{consistent read,}\\
\rho = 1-\frac{2\eta\mu}{m}+\frac{8\eta^2\tau(\tau+1)}{m^2\sqrt{p_{\min}}}+\frac{8\eta^2}{m^2p_{\min}}+\frac{32\eta^4\tau^2(\tau+1)^2}{m^4p_{\min}^2}, & \text{inconsistent read.}
\end{cases}$$

Proof By assumption, $\eta_k \equiv \eta$, $\forall k$. In Lemma 8, take

$$\beta = \begin{cases} \frac{4\eta\tau}{m\sqrt{p_{\min}}}, & \text{consistent read,}\\ \frac{4\eta\tau(\tau+1)}{m\sqrt{p_{\min}}}, & \text{inconsistent read.}\end{cases}$$

Then in (53), $a_k+b_k \equiv \rho$, $\forall k$. In addition, it is straightforward to verify $0<\rho<1$ for the specified η, and thus (54) immediately follows from Lemma 6 with $\kappa = 2\tau$ applied to (53).

4 Experiments

We illustrate the behavior of ARock for solving linear equations and ℓ_1 regularized logistic regression problems. Our primary goal is to show the efficiency of the async-parallel implementation compared to the single-threaded implementation and the sync-parallel implementation.

Our experiments run on 1 to 32 threads on a machine with eight Quad-Core AMD Opteron$^{\text{TM}}$ Processors (32 cores in total) and 64 Gigabytes of RAM. All of the experiments were coded in C++ and OpenMP. We use the Eigen library[7] for sparse matrix operations. Our codes as well as numerical results for other applications will be publicly available on the authors' website.

The running times and speedup ratios of both sync-parallel and async-parallel algorithms are sensitive to a number of factors, such as the size of each coordinate update (granularity), sparsity of the problem data, compiler optimization flags, and operations that affect cache performance and memory access contention. In addition, since all agents in the sync-parallel implementation must wait for the last agent to finish an iteration, a large load imbalance will significantly degrade the performance. We do not have the space in this paper to present numerical results under all variations of these cases.

4.1 Linear equations

We apply ARock presented in Subsection 2.1 to solve linear equations $Ax = b$ on two synthetic datasets. In Dataset I, $A \in \mathbb{R}^{n\times n}$ is a banded matrix with a small bandwidth. Given the bandwidth w, we have $a_{ij} = 0$ whenever $|j - i| > w$. In Dataset II, $A \in \mathbb{R}^{n\times n}$ is a dense random symmetric matrix. Both matrices A are diagonally dominant. They are summarized in Table 1. For each dataset, the entries of x independently follow the standard Gaussian distribution, and b is generated as Ax.

Name	Type	Size (n)	Bandwidth (w)
Dataset I	Sparse	1, 000, 000	5
Dataset II	Dense	5, 000	N/A

Table 1: Linear equation datasets

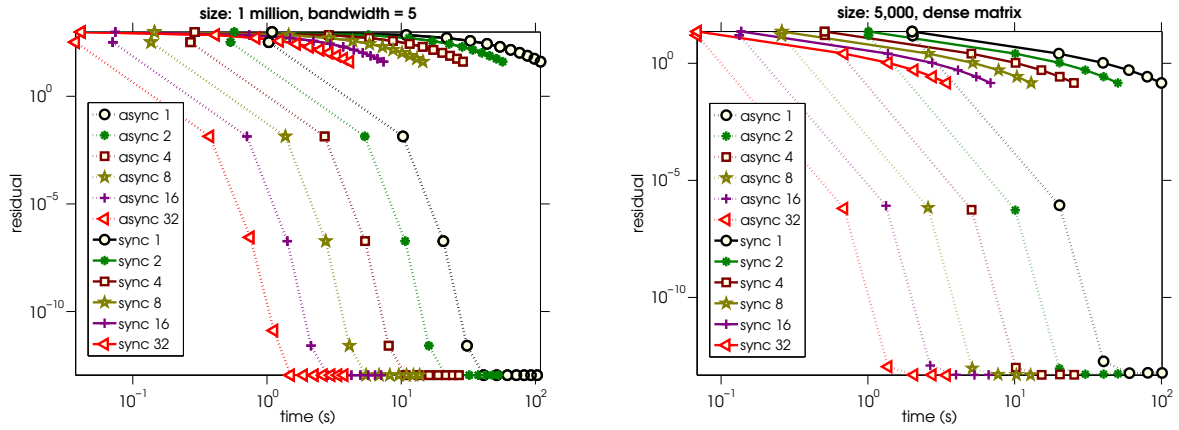
Throughout the experiments, A, b and x are shared variables. We set $\eta_k = 1$, $\forall k$. Although the theory (Theorem 3) requires a smaller step size if more cores are used (usually leading to a larger delay τ), we observe numerically that the unit step size can serve well for the algorithm on different numbers of cores. The components of x are evenly partitioned into coordinates, and each coordinate contains $\frac{n}{\text{number of cores}}$ components of x, where the number of cores varies from 1 to 32. Each core is in charge of updating one pre-assigned coordinate[8]. We run both sync-parallel Jacobi and ARock algorithms to 100 epochs on Dataset I, and to 50 epochs on Dataset II, where an epoch is counted for every n updates to the components of x.

Figure 3 depicts how the size of residual, $\|Ax - b\|$, reduces over the wall-clock time. From the figure, we see that both sync-parallel Jacobi and ARock show almost-linear speedup as the number of cores increases,

[7] http://eigen.tuxfamily.org

[8] Deterministic assignments are applied since generating pseudo-random numbers is more expensive than updating a coordinate in this test. We tested random assignments and observed that they give nearly the same epoch-versus-convergence performance but takes a much longer running time.

but ARock takes much less time than sync-parallel Jacobi. To compare their epoch convergence rates, we compute the size of residual after each epoch (though they take different times to finish an epoch). Figure 4 plots the sizes of residual at the end of different epochs. Note that the curves of sync-parallel Jacobi running on different numbers of cores are nearly identical. Hence, only one curve is shown. The curves of ARock running on different numbers of cores are also nearly identical, and they closely match with the reference curve of the standard Gauss-Seidel method, which is a sequential method. Therefore, ARock enjoys not only almost-linear speedup but also the Gauss-Seidel type of fast convergence, thanks to the asynchrony. This phenomenon was also observed in the previous work [13, 10].



(a) Dataset I (b) Dataset II

Fig. 3: Residual $\|Ax-b\|$ versus wall-clock time for sync-parallel Jacobi and ARock (async-parallel) running on 1–32 cores. All runs are terminated at 100 epochs on Dataset I and 50 epochs on Dataset II.

4.2 ℓ_1 regularized logistic regression

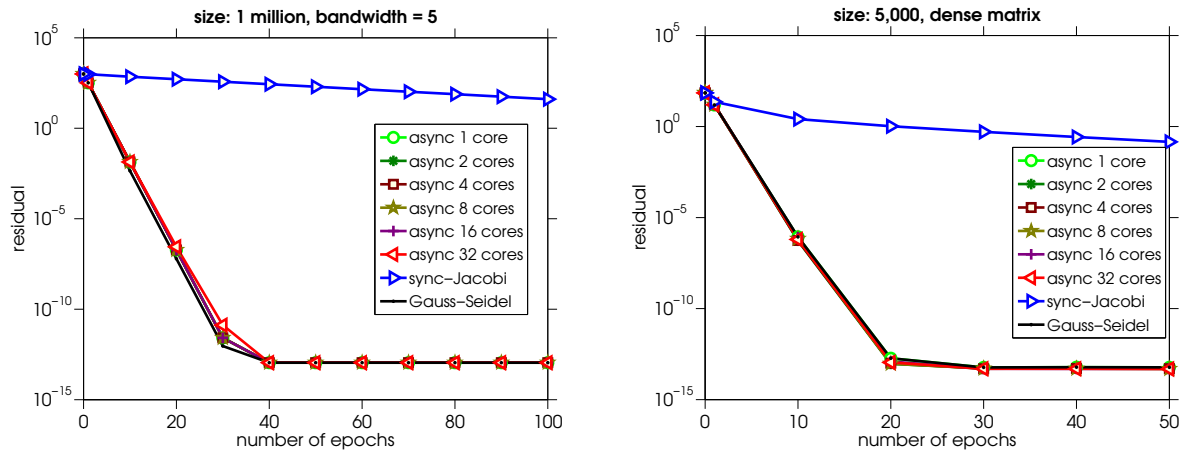
In this subsection, we apply ARock with the update (16) to the ℓ_1 regularized logistic regression problem:

$$\underset{x\in\mathbb{R}^n}{\text{minimize}}\ \lambda\|x\|_1+\frac{1}{N}\sum_{i=1}^{N}\log\left(1+\exp(-b_i\cdot a_i^T x)\right), \tag{55}$$

where $\{(a_i,b_i)\}_{i=1}^N$ is the set of sample-label pairs with $b_i\in\{1,-1\}$, $\lambda=0.0001$, and n and N represent the numbers of features and samples, respectively. This test uses the datasets[9]: rcv1 and news20, which are summarized in Table 2.

We let each coordinate hold roughly 50 features. Since the total number of features is not divisible by 50, some coordinates have 51 features. We let each agent draw a coordinate uniformly at random at each iteration. We stop all the tests after 100 epochs since they have nearly identical progress per iteration. The

[9] http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/



(a) Dataset I (b) Dataset II

Fig. 4: Epoch-convergence (regardless of time) of sync-parallel Jacobi and ARock (async-parallel). The black residual curve of the standard Gauss-Seidel method is plotted for reference.

Name	# samples	# features	# nonzeros in $\{a_1, \ldots, a_N\}$
rcv1	20, 242	47, 236	1, 498, 952
news20	19, 996	1, 355, 191	9, 097, 916

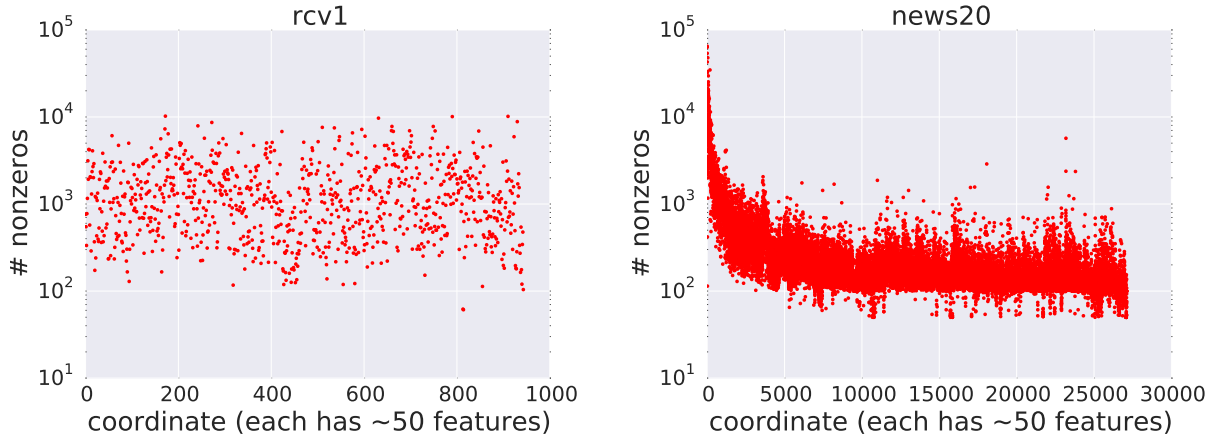
Table 2: Two datasets for sparse logistic regression

step size is set to $\eta_k = 0.9$, $\forall k$. Let $A = [a_1, \ldots, a_N]^T$ and $b = [b_1, ..., b_N]^T$. In global memory, we store A, b and x. We also store the product Ax in global memory so that the forward step can be efficiently computed. Whenever a coordinate of x gets updated, Ax is immediately updated at a low cost. Note that if Ax is *not* stored in global memory, every coordinate update will have to compute Ax from scratch, which involves the entire x and will be very expensive.

Table 3 gives the running times of the sync-parallel and ARock (async-parallel) implementations on the two datasets. We can observe that ARock achieves almost-linear speedup, but sync-parallel scales very poorly as we explain below.

In the sync-parallel implementation, all the running cores have to wait for the last core to finish an iteration, and therefore if a core has a large load, it slows down the iteration. Although every core is (randomly) assigned to roughly the same number of features (either 50 or 51 components of x) at each iteration, their a_i's have very different numbers of nonzeros (see Figure 5 for the distribution), and the core with the largest number of nonzeros is the slowest. (Sparse matrix computation is used for both datasets, which are very large.) As more cores are used, despite that they altogether do more work at each iteration, the per-iteration time reduces as the slowest core tends to be slower. The very large imbalance of load explains why the 32 cores only give speedup ratios of 4.0 and 1.3 in Table 3.

On the other hand, being asynchronous, ARock does not suffer from the load imbalance. Its performance grows nearly linear with the number of cores. In theory, a large load imbalance may cause a large maximum delay, τ, and thus a small η_k. However, the uniform $\eta_k = 0.9$ works well in all the tests, possibly because the a_i's are sparse.



(a) dataset rcv1 (b) dataset news20

Fig. 5: The distribution of coordinate sparsity. Each dot represents the total number of nonzeros in the vectors a_i that correspond to each coordinate. The large distribution in (b) is responsible for the large load imbalance and thus the poor sync-parallel performance.

Finally, we have observed that the progress toward solving (55) is mainly a function of the number of epochs and does not change appreciably when the number of cores increases or between sync-parallel and async-parallel. Therefore, we always stop at 100 epochs.

# cores	rcv1				news20			
	Time (s)		Speedup		Time (s)		Speedup	
	async	sync	async	sync	async	sync	async	sync
1	122.0	122.0	1.0	1.0	591.1	591.3	1.0	1.0
2	63.4	104.1	1.9	1.2	304.2	590.1	1.9	1.0
4	32.7	83.7	3.7	1.5	150.4	557.0	3.9	1.1
8	16.8	63.4	7.3	1.9	78.3	525.1	7.5	1.1
16	9.1	45.4	13.5	2.7	41.6	493.2	14.2	1.2
32	4.9	30.3	24.6	4.0	22.6	455.2	26.1	1.3

Table 3: Running times of ARock (async-parallel) and sync-parallel FBS implementations for ℓ_1 regularized logistic regression on two datasets. Sync-parallel has very poor speedup due to the large distribution of coordinate sparsity (Figure 5) and thus the large load imbalance across cores.

5 Conclusion

We have proposed an async-parallel framework, ARock, for finding a fixed-point of a nonexpansive operator by coordinate updates. We establish the almost sure weak and strong convergence, linear convergence rate of

ARock under certain assumptions. Preliminary numerical results on both synthetic and real data illustrate the high efficiency of the proposed framework compared to the traditional parallel (sync-parallel) algorithms.

6 Acknowledgements

We would like to thank Brent Edmunds for offering invaluable suggestion on the organization and writing of this paper. We would also like to thank Robert Hannah for coming up with the dual-memory approach. The authors are grateful to Kun Yuan for helpful discussions on decentralized optimization.

References

1. Dan Aharoni and Amnon Barak. Parallel iterative discontinuous galerkin finite-element methods. In *Discontinuous Galerkin Methods*, pages 247–254. Springer, 2000. 1.5
2. Dganit Amitai, Amir Averbuch, Moshe Israeli, and Samuel Itzikowitz. Implicit-explicit parallel asynchronous solver of parabolic pdes. *SIAM Journal on Scientific Computing*, 19(4):1366–1404, 1998. 1.5
3. Haim Avron, Alex Druinsky, and Anshul Gupta. Revisiting asynchronous linear solvers: Provable convergence rate through randomization. In *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pages 198–207. IEEE, 2014. 1.5
4. Jacques Bahi, Jean-Claude Miellou, and Karim Rhofir. Asynchronous multisplitting methods for nonlinear fixed point problems. *Numerical Algorithms*, 15(3-4):315–345, 1997. 1.5
5. Gérard M Baudet. Asynchronous iterative methods for multiprocessors. *Journal of the ACM (JACM)*, 25(2):226–244, 1978. 1.5
6. Heinz H Bauschke and Patrick L Combettes. *Convex analysis and monotone operator theory in Hilbert spaces*. Springer Science & Business Media, 2011. 1, 2.4, 3.1, 3.1
7. Didier El Baz, Andreas Frommer, and Pierre Spiteri. Asynchronous iterations with flexible communication: contracting operators. *Journal of Computational and Applied Mathematics*, 176(1):91 – 103, 2005. 1.5
8. Dimitri P Bertsekas. Distributed asynchronous computation of fixed points. *Mathematical Programming*, 27(1):107–120, 1983. 1.5
9. Dimitri P Bertsekas and John N Tsitsiklis. *Parallel and distributed computation: numerical methods*, volume 23. Prentice hall Englewood Cliffs, NJ, 1989. 1.5, 2.2
10. Dimitri P Bertsekas and John N Tsitsiklis. Some aspects of parallel and distributed iterative algorithmsa survey. *Automatica*, 27(1):3–21, 1991. 1, 4.1
11. Iain Bethune, J Mark Bull, Nicholas J Dingle, and Nicholas J Higham. Performance analysis of asynchronous jacobi's method implemented in mpi, shmem and openmp. *International Journal of High Performance Computing Applications*, 28(1):97–111, 2014. 1.5
12. Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011. 2.6.1
13. J Mark Bull and TL Freeman. *Numerical performance of an asynchronous Jacobi iteration*. Springer, 1992. 4.1
14. Kai-Wei Chang, Cho-Jui Hsieh, and Chih-Jen Lin. Coordinate descent method for large-scale l2-loss linear support vector machines. *The Journal of Machine Learning Research*, 9:1369–1398, 2008. 1.1
15. M. Chau, P. Spiteri, R. Guivarch, and H.C. Boisson. Parallel asynchronous iterations for the solution of a 3d continuous flow electrophoresis problem. *Computers & Fluids*, 37(9):1126 – 1137, 2008. 1.5
16. Daniel Chazan and Willard Miranker. Chaotic relaxation. *Linear algebra and its applications*, 2(2):199–222, 1969. 1.5
17. Patrick L Combettes and Jean-Christophe Pesquet. Stochastic quasi-fejér block-coordinate fixed point iterations with random sweeping. *arXiv preprint arXiv:1404.7536*, 2014. 3.1, 3.1, 3.1
18. Damek Davis and Wotao Yin. Convergence rate analysis of several splitting schemes. *arXiv preprint arXiv:1406.4834*, 2014. 2.5
19. Damek Davis and Wotao Yin. Faster convergence rates of relaxed peaceman-rachford and ADMM under regularity assumptions. *arXiv preprint arXiv:1407.5210*, 2014. 2.5
20. Damek Davis and Wotao Yin. A three-operator splitting scheme and its optimization applications. *arXiv preprint arXiv:1504.01032*, 2015. 1

21. Diego A Donzis and Konduri Aditya. Asynchronous finite-difference schemes for partial differential equations. *Journal of Computational Physics*, 274:370–392, 2014. 1.5
22. Didier El Baz, Didier Gazen, Mohamed Jarraya, Pierre Spiteri, and Jean Claude Miellou. Flexible communication for parallel asynchronous methods with application to a nonlinear optimization problem. *Advances in Parallel Computing*, 12:429–436, 1998. 1.5
23. Didier El Baz, Pierre Spiteri, Jean Claude Miellou, and Didier Gazen. Asynchronous iterative algorithms with flexible communication for nonlinear network flow problems. *Journal of Parallel and Distributed Computing*, 38(1):1–15, 1996. 1.5
24. Mouhamed Nabih El Tarazi. Some convergence results for asynchronous algorithms. *Numerische Mathematik*, 39(3):325–340, 1982. 1.5
25. Lei Fang and Panos J Antsaklis. Information consensus of asynchronous discrete-time multi-agent systems. In *American Control Conference, 2005. Proceedings of the 2005*, pages 1883–1888. IEEE, 2005. 1.5
26. Andreas Frommer, Hartmut Schwandt, and Daniel B Szyld. Asynchronous weighted additive schwarz methods. *Electronic Transactions on Numerical Analysis*, 5:48–61, 1997. 1.5
27. Andreas Frommer and Daniel B Szyld. On asynchronous iterations. *Journal of computational and applied mathematics*, 123(1):201–216, 2000. 1.5
28. Daniel Gabay. Chapter ix applications of the method of multipliers to variational inequalities. *Studies in mathematics and its applications*, 15:299–331, 1983. 2.6
29. Roland Glowinski and A Marroco. Sur l'approximation, par elements finis d'ordre un, et la resolution, par penalisation-dualite d'une classe de problemes de dirichlet non lineaires. *ESAIM: Mathematical Modelling and Numerical Analysis-Modélisation Mathématique et Analyse Numérique*, 9(R2):41–76, 1975. 1
30. Mingyi Hong. A distributed, asynchronous and incremental algorithm for nonconvex optimization: An admm based approach. *arXiv preprint arXiv:1412.6058*, 2014. 1.5
31. White House. Big data: Seizing opportunities, preserving values, 2014. 1
32. Cho-Jui Hsieh, Hsiang-Fu Yu, and Inderjit S Dhillon. Passcode: Parallel asynchronous stochastic dual co-ordinate descent. *arXiv preprint arXiv:1504.01365*, 2015. 1.5
33. Franck Iutzeler, Pascal Bianchi, Philippe Ciblat, and Walid Hachem. Asynchronous distributed optimization using a randomized alternating direction method of multipliers. In *Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on*, pages 3671–3676. IEEE, 2013. 1.5
34. Mark Aleksandrovich Krasnosel'skii. Two remarks on the method of successive approximations. *Uspekhi Matematicheskikh Nauk*, 10(1):123–127, 1955. 1
35. Aapo Kyrola, Daniel Bickson, Carlos Guestrin, and Joseph K Bradley. Parallel coordinate descent for l1-regularized loss minimization. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 321–328, 2011. 1.5
36. B Lang, JC Miellou, and P Spiteri. Asynchronous relaxation algorithms for optimal control problems. *Mathematics and computers in simulation*, 28(3):227–242, 1986. 1.5
37. Pierre-Louis Lions and Bertrand Mercier. Splitting algorithms for the sum of two nonlinear operators. *SIAM Journal on Numerical Analysis*, 16(6):964–979, 1979. 1, 2.5
38. Ji Liu and Stephen J Wright. Asynchronous stochastic coordinate descent: Parallelism and convergence properties. *SIAM Journal on Optimization*, 25(1):351–376, 2015. 1, 1.5
39. Ji Liu, Stephen J Wright, Christopher Ré, Victor Bittorf, and Srikrishna Sridhar. An asynchronous parallel stochastic coordinate descent algorithm. *Journal of Machine Learning Research*, 16:285–322, 2015. 1.5, 2.2
40. A Nedić, Dimitri P Bertsekas, and Vivek S Borkar. Distributed asynchronous incremental subgradient methods. *Studies in Computational Mathematics*, 8:381–407, 2001. 1.5
41. Angelia Nedic and Asuman Ozdaglar. Distributed subgradient methods for multi-agent optimization. *Automatic Control, IEEE Transactions on*, 54(1):48–61, 2009. 2.3, 2.3
42. Yu Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012. 1.4
43. Gregory B Passty. Ergodic convergence to a zero of the sum of monotone operators in hilbert space. *Journal of Mathematical Analysis and Applications*, 72(2):383–390, 1979. 1
44. Zhimin Peng, Ming Yan, and Wotao Yin. Parallel and distributed sparse optimization. In *Signals, Systems and Computers, 2013 Asilomar Conference on*, pages 659–646. IEEE, 2013. 1.5
45. Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 693–701, 2011. 1.5
46. Peter Richtárik and Martin Takáč. Parallel coordinate descent methods for big data optimization. *Mathematical Programming, Series A*, pages 1–52, 2015. 1.5
47. Herbert Robbins and David Siegmund. A convergence theorem for non negative almost supermartingales and some applications. In *Herbert Robbins Selected Papers*, pages 111–135. Springer, 1985. 3

48. Jack L Rosenfeld. A case study in programming for parallel-processors. *Communications of the ACM*, 12(12):645–655, 1969. 1.5
49. John C. Strikwerda. A probabilistic analysis of asynchronous iteration. *Linear Algebra and its Applications*, 349(13):125 – 154, 2002. 1.5
50. Xue-Cheng Tai and Paul Tseng. Convergence rate analysis of an asynchronous space decomposition method for convex minimization. *Mathematics of Computation*, 71(239):1105–1135, 2002. 1.5
51. Paul Tseng. On the rate of convergence of a partially asynchronous gradient projection algorithm. *SIAM Journal on Optimization*, 1(4):603–619, 1991. 1.5
52. Paul Tseng, Dimitri P Bertsekas, and John N Tsitsiklis. Partially asynchronous, parallel algorithms for network flow and other problems. *SIAM Journal on Control and Optimization*, 28(3):678–710, 1990. 1.5
53. Ermin Wei and Asuman Ozdaglar. On the o(1/k) convergence of asynchronous distributed alternating direction method of multipliers. In *Global Conference on Signal and Information Processing (GlobalSIP), 2013 IEEE*, pages 551–554. IEEE, 2013. 1.5, 2.6.2, B, B
54. Yangyang Xu and Wotao Yin. A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion. *SIAM Journal on imaging sciences*, 6(3):1758–1789, 2013. 1.1
55. Yangyang Xu and Wotao Yin. A globally convergent algorithm for nonconvex optimization based on block coordinate update. *arXiv preprint arXiv:1410.1386*, 2014. 1.1
56. Ming Yan and Wotao Yin. Self equivalence of the alternating direction method of multipliers. *arXiv preprint arXiv:1407.7400*, 2014. 2.5
57. Kun Yuan, Qing Ling, and Wotao Yin. On the convergence of decentralized gradient descent. *arXiv preprint arXiv:1310.7063*, 2013. 2.3
58. Ruiliang Zhang and James Kwok. Asynchronous distributed admm for consensus optimization. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1701–1709, 2014. 1.5

A Derivation of certain updates

We show in details how to obtain the updates in (20) and (25).

A.1 Derivation of updates in (20)

Let $x = (x_1, \ldots, x_m) \in \mathcal{H}^m$,

$$f(x) := \sum_{i=1}^m \mathcal{I}_{C_i}(x_i), \qquad g(x) := \mathcal{I}_{\{x_1=\cdots=x_m\}}(x),$$

where $g(x)$ equals 0 if $x_1 = \cdots = x_m$ and ∞ otherwise. Then (19a) reduces to

$$\begin{aligned}
\hat{x}^k &= \arg\min_{z\in\mathcal{H}^m} g(z) + \frac{1}{2\gamma}\|z - \hat{z}^k\|^2 \\
&= \arg\min_{z\in\mathcal{H}^m: z_1=\cdots=z_m} \|z - \hat{z}^k\|^2 \\
&= \arg\min_{z\in\mathcal{H}^m: z_1=\cdots=z_m} \sum_{i=1}^m \|z_1 - \hat{z}_i^k\|^2 \\
&= \left(\frac{1}{m}\sum_{i=1}^m \hat{z}_i^k, \ldots, \frac{1}{m}\sum_{i=1}^m \hat{z}_i^k\right) \in \mathcal{H}^m,
\end{aligned}$$

where the last equality is obtained by noting that $z_1 = \frac{1}{m}\sum_{i=1}^m \hat{z}_i^k$ is the unique minimizer of $\sum_{i=1}^m \|z_1 - \hat{z}_i^k\|^2$. Next, (19b) reduces to

$$\begin{aligned}
\hat{y}^k &= \arg\min_{z\in\mathcal{H}^m} f(z) + \frac{1}{2\gamma}\|z - (2\hat{x}^k - \hat{z}^k)\|^2 \\
&= \arg\min_{z: z_i\in C_i, \forall i} \sum_{i=1}^m \|z_i - (2\hat{x}_i^k - \hat{z}_i^k)\|^2.
\end{aligned}$$

It is easy to see that $\hat{y}_i^k = \mathrm{Proj}_{C_i}(2\hat{x}_i^k - \hat{z}_i^k)$, $\forall i$.

Since (19c) only updates the i_kth coordinate of z, we only need $\hat{x}_{i_k}^k$ and $\hat{y}_{i_k}^k$, and thus in (20a) and (20b), we only compute $\hat{x}_{i_k}^k$ and $\hat{y}_{i_k}^k$. Plugging the above $\hat{x}^k$ and $\hat{y}^k$ into (19c) gives (20c) directly.

A.2 Derivation of updates in (25)

We first get (22). The Lagrangian of (21) is

$$L(x,y,w) = f(x) + g(y) - \langle w, Ax + By - b\rangle,$$

and the Lagrange dual function is

$$\begin{aligned} d(w) &= \min_{x\in\mathcal{H}_1, y\in\mathcal{H}_2} L(x,y,w) \\ &= \left(\min_{x\in\mathcal{H}_1} f(x) - \langle A^*w, x\rangle\right) + \left(\min_{y\in\mathcal{H}_2} g(y) - \langle B^*w, y\rangle\right) + \langle w, b\rangle \\ &= -\left(\max_{x\in\mathcal{H}_1} -f(x) + \langle A^*w, x\rangle\right) - \left(\max_{y\in\mathcal{H}_2} -g(y) + \langle B^*w, y\rangle\right) + \langle w, b\rangle \\ &= -f^*(A^*w) - g^*(B^*w) + \langle w, b\rangle, \end{aligned}$$

where the last equality is from the definition of convex conjugate: $f^*(z) = \max_x \langle z, x\rangle - f(x)$. Hence, the dual problem is $\max_w d(w)$, which is equivalent to (22).

Secondly, we show why $z^+ = \mathbf{prox}_{\gamma\cdot d_g}(z)$ is given by (24). Note

$$\begin{aligned} \min_s d_g(s) + \frac{1}{2\gamma}\|s - z\|^2 &= \min_s g^*(B^*s) - \langle s, b\rangle + \frac{1}{2\gamma}\|s - z\|^2 \\ &= \min_s \max_y \langle B^*s, y\rangle - g(y) - \langle s, b\rangle + \frac{1}{2\gamma}\|s - z\|^2 \\ &= \max_y \min_s \langle B^*s, y\rangle - g(y) - \langle s, b\rangle + \frac{1}{2\gamma}\|s - z\|^2 \\ &= \max_y \min_s \langle s, By - b\rangle - g(y) + \frac{1}{2\gamma}\|s - z\|^2 \\ &= \max_y -g(y) + \langle z, By - b\rangle - \frac{\gamma}{2}\|By - b\|^2 \\ &= -\min_y g(y) - \langle z, By - b\rangle + \frac{\gamma}{2}\|By - b\|^2, \end{aligned}$$

where the fifth equality holds because

$$s^* = z - \gamma(By - b) = \arg\min_s \langle s, By - b\rangle + \frac{1}{2\gamma}\|s - z\|^2.$$

Hence, by the definition of the proximal operator and the above arguments, we have that $z^+ = \mathbf{prox}_{\gamma\cdot d_g}(z)$ can be obtained from (24). Then (23) is from (24) through replacing g to f, B to A, and b to 0.

Finally, it is straightforward to have (25) by plugging (23) and (24) into (19).

B Derivation of async-parallel ADMM for decentralized optimization

This section describes how to implement the updates (25) for the model (33).

In (33), $g(y)$ and b vanish and, corresponding to the two constraints $x_i = y_{ij}$ and $x_j = y_{ij}$, the two rows of matrices A and B are

$$\begin{bmatrix} \cdots & 1 & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & 1 & \cdots \end{bmatrix} \quad \begin{bmatrix} \cdots & -1 & \cdots \\ \cdots & -1 & \cdots \end{bmatrix},$$

where $\cdots$ are zeros, the two coefficients 1 correspond to x_i and x_j, and the two coefficients -1 correspond to y_{ij}. Then, (25a) and (25b) can be calculated as follows:

$$\begin{aligned}
\hat{y}_{li}^k &= (\hat{z}_{li,l}^k + \hat{z}_{li,i}^k)/(2\gamma) \quad \forall l \in L(i),\\
(\hat{w}_g^k)_{li,i} &= (\hat{z}_{li,i}^k - \hat{z}_{li,r}^k)/2 \quad \forall l \in L(i),\\
\hat{y}_{ir}^k &= (\hat{z}_{ir,i}^k + \hat{z}_{ir,r}^k)/(2\gamma) \quad \forall r \in R(i),\\
(\hat{w}_g^k)_{ir,i} &= (\hat{z}_{ir,i}^k - \hat{z}_{ir,r}^k)/2 \quad \forall r \in R(i).
\end{aligned}$$

In addition, $\hat{x}_i^k$ can be obtained by solving (34a), and both $z_{li,i}^{k+1}$ and $z_{ir,i}^{k+1}$ can be updated from (34b) and (34c).

Furthermore, as mentioned in Section 2.6.2, we can derive another version of async-parallel ADMM for decentralized optimization, which reduces to the algorithm in [53], by activating an edge $(i,j) \in E$ instead of an agent i each time. In this version, the agents i and j associated with the edge (i,j) must also be activated. Here we derive the update (25) for the model (33) with the update order of x and y swapped. Following (25) we obtain the following steps whenever an edge $(i,j) \in E$ is activated:

$$\begin{aligned}
\hat{x}_i^k &= \arg\min_{x_i} f_i(x_i) - \Big(\sum_{l\in L(i)} \hat{z}_{li,i}^k + \sum_{r\in R(i)} \hat{z}_{ir,i}^k\Big)x_i + \frac{\gamma}{2}|E(i)| \cdot \|x_i\|^2\\
\hat{x}_j^k &= \arg\min_{x_j} f_j(x_j) - \Big(\sum_{l\in L(j)} \hat{z}_{lj,j}^k + \sum_{r\in R(j)} \hat{z}_{jr,j}^k\Big)x_j + \frac{\gamma}{2}|E(j)| \cdot \|x_j\|^2\\
(\hat{w}_f^k)_{ij,i} &= \hat{z}_{ij,i}^k - \gamma \hat{x}_i^k\\
(\hat{w}_f^k)_{ij,j} &= \hat{z}_{ij,j}^k - \gamma \hat{x}_j^k\\
\hat{y}_{ij}^k &= \arg\min_{y_{ij}} \langle 2(\hat{w}_f^k)_{ij,i} - \hat{z}_{ij,i}^k + 2(\hat{w}_f^k)_{ij,j} - \hat{z}_{ij,j}^k, y_{ij}\rangle + \frac{\gamma}{2}\|y_{ij}\|^2\\
(\hat{w}_g^k)_{ij,i} &= 2(\hat{w}_f^k)_{ij,i} - \hat{z}_{ij,i}^k + \gamma \hat{y}_{ij}^k\\
(\hat{w}_g^k)_{ij,j} &= 2(\hat{w}_f^k)_{ij,j} - \hat{z}_{ij,j}^k + \gamma \hat{y}_{ij}^k\\
z_{ij,i}^{k+1} &= z_{ij,i}^k + \eta_k((\hat{w}_g^k)_{ij,i} - (\hat{w}_f^k)_{ij,i})\\
z_{ij,j}^{k+1} &= z_{ij,j}^k + \eta_k((\hat{w}_g^k)_{ij,j} - (\hat{w}_f^k)_{ij,j}).
\end{aligned}$$

Every agent i in the network maintains the dual variables $z_{li,i}$, $l \in L(i)$, and $z_{ir,i}$, $r \in R(i)$, and the variables x, y, w are intermediate and do not need to be maintained between the activations. When an edge (i,j) is activated, the agents i and j first compute their $\{\hat{x}_i^k, (\hat{w}_f^k)_{ij,i}\}$ and $\{\hat{x}_j^k, (\hat{w}_f^k)_{ij,j}\}$ independently and respectively, then they collaboratively compute $\hat{y}_{ij}^k$, and finally they update their own $z_{ij,i}^k$ and $z_{ij,j}^k$, respectively. We allow adjacent edges (which share agents) to be activated in a short period of time when their updates are possibly overlapped in time. When $\tau = 0$, i.e., there is no simultaneous activation or overlap, it reduces to the algorithm in [53].